

**8500
Modular MDL
Series**

**8086/8087/8088
Emulator Specifics**

This manual supports the
following TEKTRONIX products:

8300E15
8300P15
8300P16
8300P17
8300P18

Users Manual

This manual supports a software/firmware
module that is compatible with:

DOS/50 Version 2 (8550)
OS/40 Version 1 (8540)

*Please check for change information
at the rear of this manual*

LIMITED RIGHTS LEGEND

Software License No.

Contractor: Tektronix, Inc.

Explanation of Limited Rights Data Identification Method

Used: Entire document subject to limited rights.

Those portions of this technical data indicated as limited rights data shall not, without the written permission of the above Tektronix, be either (a) used, released or disclosed in whole or in part outside the Customer, (b) used in whole or in part by the Customer for manufacture or, in the case of computer software documentation, for preparing the same or similar computer software, or (c) used by a party other than the Customer, except for: (i) emergency repair or overhaul work only, by or for the Customer, where the item or process concerned is not otherwise reasonably available to enable timely performance of the work, provided that the release or disclosure hereof outside the Customer shall be made subject to a prohibition against further use, release or disclosure; or (ii) release to a foreign government, as the interest of the United States may require, only for information or evaluation within such government or for emergency repair or overhaul work by or for such government under the conditions of (i) above. This legend, together with the indications of the portions of this data which are subject to such limitations shall be included on any reproduction hereof which includes any part of the portions subject to such limitations.

RESTRICTED RIGHTS IN SOFTWARE

The software described in this document is licensed software and subject to **restricted rights**. The software may be used with the computer for which or with which it was acquired. The software may be used with a backup computer if the computer for which or with which it was acquired is inoperative. The software may be copied for archive or backup purposes. The software may be modified or combined with other software, subject to the provision that those portions of the derivative software incorporating restricted rights software are subject to the same restricted rights.

Copyright © 1982, 1983 Tektronix, Inc. All rights reserved. Contents of this publication may not be reproduced in any form without the written permission of Tektronix, Inc.

Products of Tektronix, Inc. and its subsidiaries are covered by U.S. and foreign patents and/or pending patents.

TEKTRONIX, TEK, SCOPE-MOBILE, and  are registered trademarks of Tektronix, Inc. TELEQUIPMENT is a registered trademark of Tektronix U.K. Limited.

Printed in U.S.A. Specification and price change privileges are reserved.

Section 7K

8086/8087/8088 EMULATOR SPECIFICS

	Page
Introduction	7K-1
General Information	7K-1
Terminology	7K-1
Emulator Hardware Configuration	7K-1
Microprocessors Supported	7K-2
Emulation Modes	7K-2
Clock Rate	7K-2
Symbolic Debug	7K-2
Emulator-Specific Commands, Parameters, and Displays	7K-3
Register Designators	7K-3
Byte/Word Parameter	7K-4
Operating System Commands	7K-5
AL---Allocate Physical Memory	7K-5
BK---Set or Display Breakpoint Conditions	7K-5
D, F, LO, MOV, P, and SAV---Memory Manipulation Commands	7K-6
DEAL---Deallocate Physical Memory	7K-7
DI---Disassemble Object Code into Mnemonics	7K-7
DS---Display Contents of Emulator Registers	7K-8
G---Begin Program Execution	7K-9
MAP---Set or Display Memory Map Assignments	7K-9
MEM---Specify Available User Memory	7K-9
NOMEM---Specify Unavailable User Memory	7K-10
RD---Read From Emulator I/O Port	7K-10
RESET---Reinitialize Emulator	7K-11
S---Assign Value to Register or Symbol	7K-13
SEL---Select the Emulator	7K-13
SNDP---Set Numeric Data Processor	7K-14
TRA---Control Display of Executed Instructions	7K-18
WRT---Write to Emulator I/O Port	7K-21
X---Load and Execute Program	7K-21
Real-Time Prototype Analyzer (RTPA)	7K-22
Trigger Trace Analyzer (TTA) Commands	7K-22
BUS and EVE---TTA Bus Operation Designators	7K-22
CONS---Set Consecutive Events	7K-22
DISP---Display Contents of Acquisition Memory	7K-23
Special Considerations	7K-25
The HALT Instruction	7K-25
Losing Clock Causes Emulator Fault Error	7K-25
PROM Programmer Considerations	7K-25
Emulating the Execution Bus	7K-25
Viewing the Pre-Fetch Queue	7K-25
Segment Registers	7K-26
Service Calls	7K-27

Jumpers	7K-32
Emulator Timing	7K-42
Installing Your 808x Emulator Software	7K-48
8086 Demonstration Run	7K-51
Introduction	7K-51
Examine the Demonstration Program	7K-54
Assemble and Load the Demonstration Program	7K-55
Case 1: Assemble and Load on the 8550	7K-56
Case 2: Assemble on the 8560; Download to the 8540	7K-61
Case 3: Download from Your Host to the 8540	7K-67
Case 4: Patch the Program into Memory	7K-70
Run the Demonstration Program	7K-73
Monitor Program Execution	7K-78
Summary of 8086 Emulator Demonstration Run	7K-85

ILLUSTRATIONS

Fig. No.		Page
7K-1	8086/8088 FLAGS register	7K-3
7K-2	808x SVC demonstration program listing	7K-29
7K-3	8086 MX Mode timing diagram	7K-44
7K-4	8086 MN Mode timing diagram	7K-47
7K-5	8086 demonstration run program	7K-52
7K-6	8086 demonstration run program: Extended Tekhex format	7K-53
7K-7	Host computer commands for preparing demonstration program	7K-67

TABLES

Table No.		Page
7K-1	Emulator Configuration Terminology	7K-1
7K-2	8086/8088 Registers and Flags	7K-4
7K-3	8087 Symbols Set With the SNDP Command	7K-15
7K-4	808x Bus Operation Designators	7K-22
7K-5	808x Service Calls	7K-28
7K-6	P7076, P7077 and P7078 Interactions	7K-37
7K-7	Dummy Address Jumpers	7K-38
7K-8	Representative 808x Emulator/Microprocessor Timing Differences (MX Mode)	7K-42
7K-9	Representative 808x Emulator/Microprocessor Timing Differences (MN Mode)	7K-45
7K-10	Basic 8560 Editing Commands	7K-63

Section 7K

8086/8087/8088 EMULATOR SPECIFICS

INTRODUCTION

This supplement is designed to be inserted into Section 7 of the 8550 System Users Manual (DOS/50 Version 2) or the 8540 System Users Manual. This Emulator Specifics section explains the features of the 8550 and 8540 systems that are unique to the 8086, 8088, 8086/8087, and 8088/8087 emulators. Throughout this supplement, the phrase "your System Users Manual" refers to your 8550 System Users Manual or 8540 System Users Manual. The 8086 demonstration run at the end of this supplement is designed to be used with Section 1, the Learning Guide of your System Users Manual; the rest of this supplement contains reference material.

GENERAL INFORMATIONTerminology

The 8086/8088 Emulator Processor boards can be used with any of four prototype control probes. The various configurations, and the terms used to describe them throughout this Emulator Specifics supplement are shown in Table 7K-1.

Table 7K-1
Emulator Configuration Terminology

Terminology	Prototype Control Probe(s)
=====	
8086 emulator	8086
8088 emulator	8088
8086/8087 emulator	8086/8087
8088/8087 emulator	8088/8087
8086/8088 emulator	8086 or 8088
8087 emulator	8086/8087 or 8088/8087
808x emulator	any 8086, 8087 or 8088

Emulator Hardware Configuration

In emulation mode 0, the prototype control probe is connected to the emulator your program runs in 8550 or 8540 program memory. In modes 1 and 2, the prototype control probe must be connected to both the emulator and your prototype. For instructions on installing your emulator and probe, refer to your 8086/8088 Emulator Processor and Prototype Control Probe Installation Service Manual.

Microprocessors Supported

The 8086 emulator emulates the Intel 8086, 8086-2 and 8086-4 microprocessors.

The 8088 emulator emulates the Intel 8088 microprocessor. 8088 Prototype Control Probes with serial numbers B03xxxx and higher support the 8088-2 (8 MHz) microprocessor. (8088 Prototype Control Probes with serial numbers below B03xxxx can be upgraded to 8 MHz. Contact your Tektronix Service Representative for details.)

The 8086/8087 emulator emulates the 8086 microprocessors when used with the 5 MHz version of the 8087.

The 8088/8087 emulator emulates the 8088 microprocessors when used with the 5 MHz version of the 8087.

The 8086 and 8088 processors differ only in their data bus width: the 8086 uses either a 16-bit or an 8-bit data bus width, and the 8088 uses an 8-bit data bus width only.

Emulation Modes

The 808x emulators support emulation modes 0, 1, and 2, as described in the Emulation section of your System Users Manual. The 808x emulators support service calls (SVCs) in all three modes.

Clock Rate

8086/8088 Emulator. The input frequency of the 8086/8088 emulator clock (used in emulation mode 0) is 8MHz. The maximum input frequency of the prototype's clock (used in modes 1 and 2) is 8MHz.

8087 Emulator. The input frequency of the 8087 emulator clock is 5MHz. The maximum input frequency of the prototype's clock is 5MHz.

Symbolic Debug

The 808x emulators support the use of symbolic debug. Many displays in this supplement include symbolic debug information. Refer to the Emulation section of your System Users Manual for information on symbolic debug.

EMULATOR-SPECIFIC COMMANDS, PARAMETERS, AND DISPLAYSRegister Designators

Figure 7K-1 shows the contents of the 8086/8088 FLAGS register.

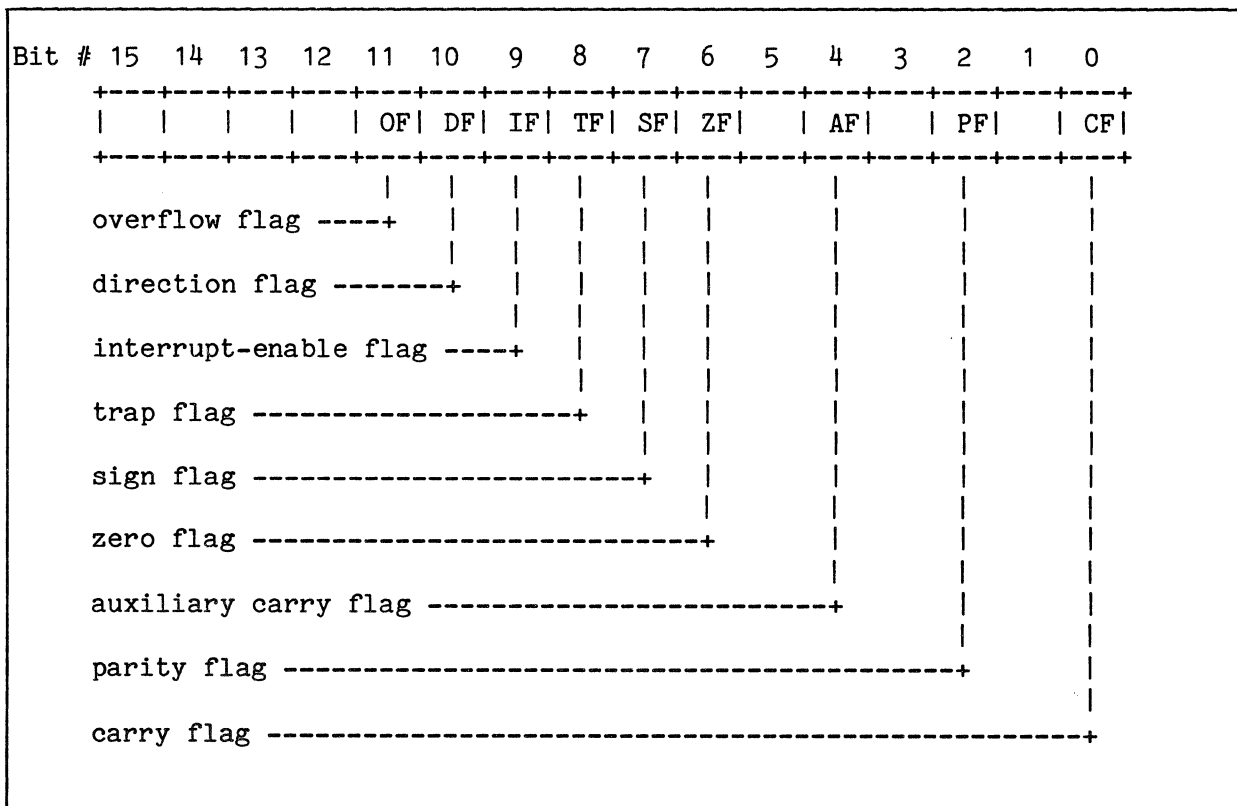


Fig. 7K-1. 8086/8088 FLAGS register.

Table 7K-2 alphabetically lists the symbols used by DOS/50 and OS/40 to designate the registers and flags used by the 808x emulators. The table provides the following information for each symbol:

- a description of the register or flag that the symbol represents
- the size of the register or flag
- the value assigned to the register or flag by the RESET command
- whether the register or flag can be assigned a value by the S (Set) command.

Table 7K-2
8086/8088 Registers and Flags

Symbol	Description	Size in Bits	Value After RESET(*a)	Altered by S Command?
AF	Auxiliary carry flag (*b)	1	0	yes
AH	High-order byte of register A	8	NC	yes
AL	Low-order byte of register A	8	NC	yes
AX	Register A	16	NC	yes
BH	High-order byte of register B	8	NC	yes
BL	Low-order byte of register B	8	NC	yes
BP	Base pointer register	16	NC	yes
BX	Register B	16	NC	yes
CF	Carry flag (*b)	1	0	yes
CH	High-order byte of register C	8	NC	yes
CL	Low-order byte of register C	8	NC	yes
CS	Code segment register	16	FFFF	yes
CX	Register C	16	NC	yes
DF	Direction flag (*b)	1	0	yes
DH	High-order byte of register D	8	NC	yes
DI	Destination index register	16	NC	yes
DL	Low-order byte of register D	8	NC	yes
DS	Data segment register	16	0000	yes
DX	Register D	16	NC	yes
ES	Extra segment register	16	0000	yes
FLAGS	Flags register (*b)	16	0000	yes
IF	Interrupt-enable flag (*b)	1	0	yes
INTR (*c)	Interrupt request input	1	NC	no
IP	Instruction pointer	16	0000	no
OF	Overflow flag (*b)	1	0	yes
NMI (*c)	Non-maskable interrupt input	1	NC	no
PF	Parity flag (*b)	1	0	yes
SF	Sign flag (*b)	1	0	yes
SI	Source index register	16	NC	yes
SP	Stack pointer register	16	NC	yes
SS	Stack segment register	16	0000	yes
TEST (*c)	Test control for WAIT instruction	1	NC	no
TF	Trap flag (*b)	1	0	yes
ZF	Zero flag (*b)	1	0	yes

(*a) NC = not changed by RESET.

(*b) The FLAGS register is illustrated in Fig. 7K-1.

(*c) INTR, NMI, and TEST are hardware inputs to the microprocessor.

Byte/Word Parameter

Several commands offer you the choice of operating on memory on a byte-oriented or word-oriented basis. In affected commands, this choice is represented by the -B or -W parameter. For the 808x emulator, the default value is -B (Byte).

OPERATING SYSTEM COMMANDS

AL---Allocate Physical Memory

The 808x emulators support the use of the AL (ALlocate) command. However, memory segment information is not valid. Use the S (Set) command to assign appropriate values to the segment registers.

The AL command allocates 4K-byte blocks of program memory to logical addresses used by your program. The 808x emulators support the -F and -S dash modifiers, which specify fast or slow memory, respectively. The default condition is fast memory (i.e., no wait states are inserted during program memory accesses). The -S modifier is used when jumpers P3505 and P7005 have been set to support it. These jumpers are discussed in the "Jumpers" subsection later in this supplement. Deallocating memory does **not** remove the "slow" designation: the -S modifier remains in effect until changed by a -F modifier.

Example. The following commands allocate memory and display the allocations:

```
> AL 0 3FFF <CR>
  4 BLOCK(S) ALLOCATED   12 BLOCK(S) FREE

> AL 8000 -S <CR>
  1 BLOCK(S) ALLOCATED   11 BLOCK(S) FREE

> AL <CR>
  00000 - 03FFF
  08000 - 08FFF S

  5 BLOCK(S) ALLOCATED   11 BLOCK(S) FREE
```

BK---Set or Display Breakpoint Conditions

The 808x emulators allow you to set up to three breakpoints. These emulators support "arming" of breakpoints with the -A dash modifier. (When breakpoints are armed, the breakpoint conditions must occur in sequence. The actual break, and its trace line, occur only when the final breakpoint condition is met.)

You can program any of three conditions: BK1 to arm BK2; BK2 to arm BK3; or BK1 to arm BK2 to arm BK3. To use this arming feature, first program BK1 (for example), then program BK2 and include the -A parameter. To clear the arming feature, you must clear the breakpoint that contains the -A parameter.

The 808x emulators also allow you to specify Read (RD), Write (WT) Memory (M) and I/O operations with the BK command.

Example. The following breakpoints set the emulator to break on any I/O read that follows a memory write to address 1C0F0

```
> BK 1 1C0F0 M WT <CR>
> BK 2 , , I RD -A <CR>
```

NOTE

Addresses 8, 9, A, and B are the vector that points to an NMI (Non-Maskable Interrupt). These addresses are used by the operating system whenever a break is generated. If you set a breakpoint at one of these addresses (for example, to test your program's NMI routine), **any** system or user interrupt that occurs will generate a break, not just an NMI.

D, F, LO, MOV, P, and SAV---Memory Manipulation Commands

With the D (Display) command, and with the other commands that manipulate memory (F, LO, MOV, P, SAV), the **loadaddr** parameter may be either an absolute address or an address relative to the current Code Segment (CS). For example, if CS=0100, the following commands are equivalent:

```
> D 1020 <CR>
    0 1 2 3 4 5 6 7 8 9 A B C D E F
001020 12 34 56 78 90 00 00 00 00 00 00 00 00 00 00 00 .4Vx.....

> D CS:20 <CR>
    0 1 2 3 4 5 6 7 8 9 A B C D E F
000020 12 34 56 78 90 00 00 00 00 00 00 00 00 00 00 00 .4Vx.....
```

NOTE

When you specify an address relative to the current CS, the display includes only the offset value and does not include the CS information.

DEAL---Deallocate Physical Memory

The DEAL (DEALlocate) command removes the previous allocation of program memory. DEAL operates on 4K-byte blocks of program memory. The -A command modifier deallocates all of program memory. Memory segment information is not valid with the DEAL command. The DEAL command does **not** affect the conditions set with the AL command modifiers -F and -S.

Example. The following commands are an example of AL and DEAL usage:

```
> AL 0 3FFF -S <CR>
  4 BLOCK(S) ALLOCATED      12 BLOCK(S) FREE

> DEAL -A <CR>

> AL 0 7FFF <CR>
  8 BLOCK(S) ALLOCATED      8 BLOCK(S) FREE

> AL <CR>
  00000 - 03FFF   S
  04000 - 07FFF

  8 BLOCK(S) ALLOCATED      8 BLOCK(S) FREE
```

DI---Disassemble Object Code into Mnemonics

The DI (DISassemble) command translates object code in memory into assembly language instructions. DI displays object code, assembly language mnemonics, and operands. Here is an example of an 808x emulator's DI output:

```
> DI 100 112 <CR>
LOC      INST
000100   BB0005
000103   B90500
000106   32C0
000108   0207
00010A   43
00010B   E2FB
00010D   BAF7FF
000110   EE
000111   90
000112   90
```

MNEM	OPER
MOVW	BX, #0500
MOVW	CX, #0005
XORB	AL, AL
ADDB	AL, [BX]
INC	BX
LOOP	\$-03
MOVW	DX, FFF7
OUT	DX, AL
NOP	
NOP	

operand(s): address, register,
or data being operated on

instruction mnemonic

hexadecimal representation of instruction

absolute address of the instruction

DS---Display Contents of Emulator Registers

The DS command displays the status and register contents of the emulator. All numbers in the DS display are hexadecimal. Here is an example of a DS display produced by an 8086/8088 emulator:

> DS <CR>

PC/CS:IP	HD STATUS/CLOCK	AX/BX	CX/DX	SI/DI	SP/BP	SS/DS	ES/F
000106	NMI=0 INTR=0 TEST=1	000F	0002	0000	0000	0000	0000
0000:0106	5MHZ	0502	FFF7	0000	0000	0000	F006

FLAGS	.	.	.	.	OF	DF	IF	TF	SF	ZF	.	AF	.	PF	.	CF
F006	X	X	X	X	0	0	0	0	0	0	X	0	X	1	X	0

When you are using an 8087 emulator, the long form of the DS display (DS -L) displays the registers and flags of the 8087 as well as the information for the 8086 or 8088. You may change the format of the 8087 portion of the display using the SNDP command described later in this supplement. Here is an example of a DS display produced by an 8087 emulator:

> DS -L <CR>

PC/CS:IP	HD STATUS/CLOCK	AX/BX	CX/DX	SI/DI	SP/BP	SS/DS	ES/F
001000	NMI=1 INTR=1 TEST=1	000F	0002	0000	0000	0000	0000
0000:1000	5MHZ	0502	FFF7	0000	0000	0000	F006

FLAGS	.	.	.	.	OF	DF	IF	TF	SF	ZF	.	AF	.	PF	.	CF
0000	X	X	X	X	0	0	0	0	0	0	X	0	X	1	X	0

CW	SW	TW	IP	OPC	OPRP
03FF	4100	FFFF	FFFFFF	D800	00000

CW	.	.	.	IC	RC	RC	PC	PC	IEM	.	PM	UM	OM	ZM	DM	IM
03FF	X	X	X	0	0	0	1	1	1	X	1	1	1	1	1	1

SW	B	C3	ST	ST	ST	C2	C1	CO	IR	.	PE	UE	OE	ZE	DE	IE
03FF	0	1	0	0	0	0	0	1	0	X	0	0	0	0	0	0

REG	WI	LSB	T	EF
ST0		0000	3	00
ST1		0000	3	00
ST2		0000	3	00
ST3		0000	3	00
ST4		0000	3	00
ST5		0000	3	00
ST6		0000	3	00
ST7		0000	3	00

Table 7K-2 explains the symbols displayed by the short form of the DS command. Refer to the description of the SNDP command for descriptions of the symbols used in the 8087 portion of the DS -L command.

G---Begin Program Execution

The G (Go) command begins program execution. With the 808x emulators, the G command operates differently than with other emulators: Usually, if you enter a command like G 1000, the program counter is set to 1000, and execution begins from there. With the the 808x emulators, the G command uses information from the Code Segment (CS) register and Instruction Pointer (IP) to calculate the effective starting address (EA). The EA is calculated as follows:

$$EA = IP + (CS*16)$$

where IP is the value of the **address** parameter you enter with the G command. Since the value of the CS register may be changed during program execution, it is recommended that you set the CS register (using the S command) before program execution, to ensure that the program starts at the desired address.

NOTE

The value of the **address** parameter of the G command must be less than 10000H.

MAP---Set or Display Memory Map Assignments

The 808x emulators support the MAP command as described in your System Users Manual, with the following exceptions:

- The 808x emulators map memory in 4K-byte blocks.
- The -M and -A modifiers are not supported.
- You may not include memory segment information in the MAP command.

MEM---Specify Available User Memory

The MEM (MEMory) command informs the emulator that the prototype contains memory at a given block of addresses. The MEM command operates on 4K-byte blocks. MEM can also be used to reverse a previous NOMEM command. (The default condition is that all prototype memory is available to program.) Memory segment information is not valid with the MEM command. For an example of MEM command usage, see the discussion of the NOMEM command.

NOMEM---Specify Unavailable User Memory

The NOMEM command informs the emulator that the prototype does **not** contain memory at a given block of addresses. (The default condition is that all prototype memory is available to the program.) The NOMEM command operates on 4K-byte blocks. Memory segment information is not valid with the NOMEM command.

Example. The following commands are an example of MEM and NOMEM usage.

```
> NOMEM 0 7FFFF <CR>
```

```
> NOMEM <CR>
INVALID USER MEMORY
00000 - 7FFFF
```

```
> MEM 0 <CR>
```

```
> MEM <CR>
VALID USER MEMORY
00000 - 00FFF
80000 - FFFFF
```

```
> NOMEM <CR>
INVALID USER MEMORY
01000 - 7FFFF
```

RD---Read From Emulator I/O Port

The RD (Read) command reads data from a prototype I/O port. This command supports the -B (Byte) and -W (Word) command modifiers. The default is -B. You may include symbolic names as input to the RD command.

The 808x emulators support both fixed-port and memory-mapped I/O. The default is fixed-port I/O. The I/O ports must be in the range 0--FFFFH. Memory-mapped I/O is selected with the -M command modifier. The 808x emulators do **not** support the -S modifier of the RD command.

NOTE

The RD command **always** reads from the prototype. In mode 0, the command internally changes to mode 1 to execute the command and then returns to mode 0. An error message occurs if the probe is not connected to the prototype when this command is executed.

RESET---Reinitialize Emulator

The RESET command sends a hardware reset signal to the emulating microprocessor and clears any pending interrupts. The "Value After RESET" column of Table 7K-2 indicates which 8086/8088 registers are affected by the RESET command.

8086/8088 Example. Suppose the DS command returns the following status:

```
> DS <CR>
PC/CS:IP      HD STATUS/CLOCK      AX/BX CX/DX SI/DI SP/BP SS/DS ES/F
000106        NMI=0 INTR=0 TEST=1    000F 0002 0000 0C30 1034 0100
0000:0106      5MHZ                  0502 FFF7 1223 0000 0FF0 F006

          FLAGS  . . . .  OF DF IF TF  SF ZF . AF  . PF . CF
          F006  X X X X   0 0 0 0   0 0 X 0   X 1 X 0
```

Enter the RESET command; then use DS to check the results.

```
> RESET <CR>
> DS <CR>
```

```

↓ ↓ ↓
PC/CS:IP      HD STATUS/CLOCK      AX/BX CX/DX SI/DI SP/BP SS/DS ES/F
000000        NMI=0 INTR=0 TEST=1    000F 0002 0000 0C30 0000 0000
FFFF:0000      5MHZ                  0502 FFF7 1223 0000 0000 0000

          FLAGS  . . . .  OF DF IF TF  SF ZF . AF  . PF . CF
          0000  X X X X   0 0 0 0   0 0 X 0   X 0 X 0
          ↑↑↑↑      ↑ ↑ ↑ ↑      ↑ ↑      ↑      ↑      ↑

```

The arrows show the changed registers.

8087 Example. Suppose the DS -L command returns the following status:

> DS -L <CR>

PC/CS:IP	HD STATUS/CLOCK	AX/BX	CX/DX	SI/DI	SP/BP	SS/DS	ES/F
000000	NMI=1 INTR=1 TEST=1	0000	0000	0000	0000	0000	0000
0000:0000	5MHZ	0000	0000	0000	0000	0000	0000

FLAGS	.	.	.	.	OF	DF	IF	TF	SF	ZF	.	AF	.	PF	.	CF
0000	X	X	X	X	0	0	0	0	0	0	X	0	X	0	X	0

CW	SW	TW	IP	OPC	OPRP
FE00	BEFF	0000	FFFFFF	DFFF	FFFFF

CW	.	.	.	IC	RC	RC	PC	PC	IEM	.	PM	UM	OM	ZM	DM	IM
FE00	X	X	X	1	1	1	1	0	0	X	0	0	0	0	0	0

SW	B	C3	ST	ST	ST	C2	C1	CO	IR	.	PE	UE	OE	ZE	DE	IE
BEFF	1	0	1	1	1	1	1	0	1	X	1	1	1	1	1	1

REG	LR	LSB	T	EF
ST0		+1.1000000000000000E+3	0	20
ST1		+2.2000000000000000E+2	0	20
ST2		+3.3000000000000000E+3	0	20
ST3		+4.4000000000000000E+4	0	20
ST4		+5.5000000000000000E+5	0	20
ST5		+6.6000000000000000E+6	0	20
ST6		+7.7000000000000000E+7	0	20
ST7		+8.8000000000000000E+8	0	20

Enter the RESET command; then use DS -L to check the results.

> RESET <CR>

> DS -L <CR>

PC/CS:IP	HD STATUS/CLOCK										AX/BX	CX/DX	SI/DI	SP/BP	SS/DS	ES/F
000000	NMI=1 INTR=1 TEST=1										0000	0000	0000	0000	0000	0000
FFFF:0000	5MHZ										0000	0000	0000	0000	0000	0000
FLAGS	.	.	.	.	OF	DF	IF	TF	SF	ZF	.	AF	.	PF	.	CF
0000	X	X	X	X	0	0	0	0	0	0	X	0	X	0	X	0

The arrows show the changed 8087 registers. CW, SW, TW, and the stack registers are the only valid reset values for the 8087. IP, OPC, and OPRP have changed, but may not be correct. The registers changed for the 8086/8088 are the same as those shown in the previous example.

S---Assign Value to Register or Symbol

The S (Set) command changes the values of the 8086 or 8088 registers. The symbols allowed and the registers they represent are shown in Table 7K-2. Refer also to the description of the SNDP command, which allows you to set registers for the 8087 microprocessor.

SEL---Select the Emulator

Enter the following command to select an 808x emulator and the 8086/8088 assembler on the 8550, or an 808x emulator on the 8540:

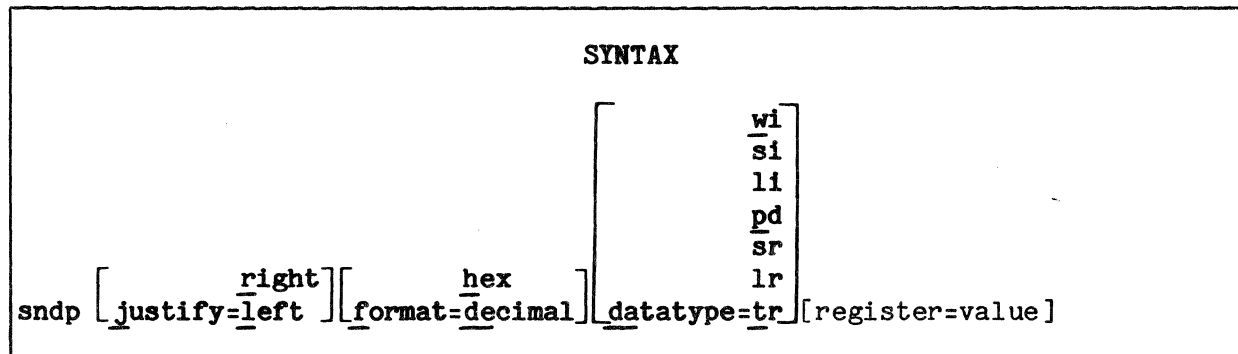
```
> SEL 8086 <CR>
```

To select the 8086/8088 assembler on the 8560, enter:

```
$ uP=8086; export uP <CR>
```

SNDP---Set Numeric Data Processor

With the 8087 emulators, an additional operating system command is provided. This command allows you to set the 8087 register values, and to change the format of the 8087 displays shown by the DS -L and TRA -L commands. The following paragraphs describe the syntax and usage of the SNDP command. Refer the Command Dictionary in your System Users Manual for an explanation of syntax and notation conventions.

**PARAMETERS**

No parameters	Displays the 8087 display options currently selected.
justify	Justifies the stack registers (REG) with the least-significant word starting either on the right, or on the left. When the format is decimal, the stack registers are always justified on the right.
format	Specifies the desired format for the stack registers. There are two options: decimal , or hex (hexadecimal).
datatype	You may specify one of seven stack register data types: wi--word integer si--short integer li--long integer pd--packed decimal sr--short real lr--long real tr--temporary real
register	The symbol of the register that you want to set.
value	The value to which you are setting the register. For the eight stack registers, this value must be consistent with the format and type that you have set. For all other registers, fields and flags, you may use an expression. Refer to the explanation for format information.

EXPLANATION

The SNDP command allows you to set eight stack registers, six miscellaneous registers, and twenty fields and flags of the 8087 microprocessor. These are listed in Table 7K-3. The SNDP command also lets you change the display format of the 8087 stack register values.

Table 7K-3
8087 Symbols Set With the SNDP Command

Symbol	Description
CW	Control word
DM	Denormalized operand mask
IC	Infinity control
IEM	Interrupt enable mask
IM	Invalid operation mask
OM	Overflow mask
PC	Precision control
PM	Precision mask
RC	Rounding control
UM	Underflow mask
ZM	Zero-divide mask
IP	Instruction address pointer
OPC	Opcode instruction
OPRP	Operand address pointer
TW	Tag word
SW	Status word
B	Busy flag
CC	Condition code
DE	Denormalized exception flag
IE	Invalid operation exception flag
IR	Interrupt request
OE	Overflow exception flag
PE	Precision exception flag
ST	Stack top pointer
UE	Underflow exception flag
ZE	Zero-divide exception flag

Value Formats. When you set the values of the register symbols, their formats must be as follows:

- The maximum number of hexadecimal digits allowed for each data type is:
 - WI--word integer, 4
 - SI--short integer, 8
 - LI--long integer, 16
 - PD--packed decimal, 20
 - SR--short real, 8
 - LR--long real, 16
 - TR--temporary real, 20
- Values start with the most significant bit (MSB) and follow the 8087 data types as they are stored in memory.
- If the number of digits entered is less than the value's data type maximum, the value is padded with leading zeroes. The MSB is set if the value has a minus sign.
- You may use a valid expression for a value except when setting the stack registers. Numeric values must be used to set the stack registers.
- Values must have a format consistent with their data type.

For decimal integers:

[+]digit(s)

For decimal floating point values:

```
[-]                [e][-]
[+]digit(s).digit(s)[E][+]digit(s)
```

For hexadecimal values:

```
[-]  
[+]hexdigit(s)
```

Other Symbols. Several other symbols are used as headings in the 8087 displays. They are shown in the following example of the 8087 portion of a TRA -L display:

		CW	SW	TW	IP	OPC	OPRP
		03FF	7320	0FFF	00112	D8C9	00500
REG	LI					LSB	T EF
ST0						+0000000000000031416	0 00
ST1						+0000000000000010000	0 00
ST2						+0000000000000000000	3 00
ST3						+0000000000000000000	3 00
ST4						+0000000000000000000	3 00
ST5						+0000000000000000000	3 00
ST6						+0000000000000000000	3 00
ST7						+0000000000000000000	3 00

REG is the heading for the list of stack registers, ST0 through ST7.

LI indicates that the data is in long integer format.

LSB shows where the least-significant bit is, and shows whether the display is justified on the right or on the left. This display is right-justified.

T is the heading for the individual tag values for each register. The values are derived from the tag word and the stack pointer.

The EF values are indications of the emulator software status. The values refer to exception flags in the Status Word. The emulator software uses the 8087 to convert the stack registers from one format to another. If any errors occur during this conversion, they cause the EF values to be something other than 0. For example, conversion errors will occur if the value in the stack register is too large for the data type you have specified.

The symbols CW, SW, TW, IP, OPC and OPRP are explained in Table 7K-3.

Stack Register Justification. The stack register values may be justified on either the left or the right, by words. For example, if your format is SI and you specified justification on the left, the first two lines of the stack register display might look something like this:

REG	SI	LSB		T	EF
ST0		1234	5678	0	00

If you then changed justification to the right, these two lines would now look like this:

REG	SI		LSB	T	EF
ST0		5678	1234	0	00

NOTE

The SNDP command can only be used with an 8087 emulator. In addition, the 8087 microprocessor must reside in the prototype control probe, and not in the prototype.

TRA---Control Display of Executed Instructions

The TRA (TRAcE) command selects the range and type of instructions to be displayed as your program executes. The short form of the trace command (TRA) displays register values for the 8086/8088 microprocessors. The long form of the trace command (TRA -L) also displays register values for the 8087, if an 8087 emulator is installed and the emulator is executing an 8087 instruction.

NOTE

Segment registers are not allowed in the TRA address parameters, because these registers may be changed during program execution. If you enter a segment register as part of a TRA parameter, an error will occur when you enter the G (Go) command.

Here is a sample 8086/8088 trace display with symbolic debug on:

> TRA ALL <CR>

> G START <CR>

PC/CS:IP	MNEMONIC/DATA	AX/BX	CX/DX	SI/DI	SP/BP	SS/DS	ES/F
START							
000100	MOVW BX,#0500	0063	0000	0000	0000	0000	0000
0000:0103	BB0005	0500	FFF7	0000	0000	0000	F006
DEMO+000103							
000103	MOVW CX,#0005	0063	0005	0000	0000	0000	0000
0000:0106	B90500	0500	FFF7	0000	0000	0000	F006
DEMO+000106							
000106	XORB AL,AL	0000	0005	0000	0000	0000	0000
0000:0108	32C0	0500	FFF7	0000	0000	0000	F046
ALOOP							
000108	ADDB AL,[BX]	0000	0005	0000	0000	0000	0000
0000:010A	0207	0500	FFF7	0000	0000	0000	F046
DEMO+00010A							
00010A	INC BX	0000	0005	0000	0000	0000	0000
0000:010B	43	0501	FFF7	0000	0000	0000	F002

8086/8088 registers

operand field

mnemonic; hexadecimal representation of the instruction

symbolic label; program counter; Code Segment register, and Instruction Pointer

NOTE

On the TRA display, "PC" points to the **last** instruction executed, while "CS:IP" points to the **next** instruction to be executed. On the DS display, both point to the next instruction.

Here is an example of a TRA -L for the 8087 emulators:

> TRA ALL -L <CR>

> G 100 <CR>

PC/CS:IP	MNEMONIC/DATA	AX/BX	CX/DX	SI/DI	SP/BP	SS/DS	ES/F
000100	MOVW AX,#2710	2710	0000	0000	0000	0000	0000
0000:0103	B81027	0000	0000	0000	0000	0000	F002
000103	MOVW 0500,AX	2710	0000	0000	0000	0000	0000
0000:0106	A30005	0000	0000	0000	0000	0000	F002
000107	FILDW 0500	2710	0000	0000	0000	0000	0000
0000:010B	DF060005	0000	0000	0000	0000	0000	F002
		CW	SW	TW	IP	OPC	OPRP
		03FF	7900	3FFF	00107	DF06	00500

REG	LI	LSB	T	EF
ST0		+00000000000000000000000000000000	0	00
ST1		+00000000000000000000000000000000	3	00
ST2		+00000000000000000000000000000000	3	00
ST3		+00000000000000000000000000000000	3	00
ST4		+00000000000000000000000000000000	3	00
ST5		+00000000000000000000000000000000	3	00
ST6		+00000000000000000000000000000000	3	00
ST7		+00000000000000000000000000000000	3	00

The headings and symbols used in the 8087 portion of the display are described under the SNDF command in this supplement.

NOTE

Instructions whose execution cannot be broken with a Non-Maskable Interrupt (NMI) are not traced during TRA ALL. These instructions include:

- Instructions that change the contents of a segment register.
- A WAIT instruction when the TST line is asserted.

WRT---Write to Emulator I/O Port

The WRT (WRiTe) command writes data to an I/O port. The WRT command supports the -B (Byte) and -W (Word) command modifiers. The default value is -B. You may include symbolic names as input to the WRT command.

The 808x emulators support both fixed-port and memory-mapped I/O. The I/O ports must be in the range 0--FFFFH. Memory-mapped I/O is selected with the -M command modifier. The 808x emulators do not support the -S modifier of the WRT command. The default condition is fixed-port I/O.

NOTE

The WRT command **always** writes to the prototype. In mode 0, the command internally changes to mode 1 to execute the command and then returns to mode 0. An error message occurs if the probe is not connected to the prototype when you enter this command.

X---Load and Execute Program

The X (eXecute) command loads a file and executes a G (Go) command at the transfer address. On the 808x emulators, the X command is supported **only** under the following conditions:

- the CS register is set to zero; and
- the transfer address is within the range 0--FFFF.

Since the transfer address is an absolute address containing both CS (Code Segment) and IP (Instruction Pointer) information, the X command can rarely be used with the 808x emulators.

Refer to the "Special Considerations" subsection for more information on program execution.

REAL-TIME PROTOTYPE ANALYZER (RTPA)

The 808x emulators do not support the Real-Time Prototype Analyzer (RTPA). The Trigger Trace Analyzer (TTA) is used instead of the RTPA.

TRIGGER TRACE ANALYZER (TTA) COMMANDS

The following text discusses the emulator-specific commands of the Trigger Trace Analyzer (TTA) option. For more information about the TTA, refer to the Trigger Trace Analyzer Users Manual, or to the TTA discussion in the Emulation section of your System Users Manual.

BUS and EVE---TTA Bus Operation Designators

Table 7K-4 lists the 808x bus operation designators recognized by the BUS command and by the -B parameter of the EVE command.

Table 7K-4
808x Bus Operation Designators

Symbol	Bus Operation Type
BHE	Bus high enable (8086)
CLR	All types
CS	Code segment
DS	Data segment
ES	Extra segment
F	First fetch ("M1")
FS	Subsequent fetches
IA	Interrupt acknowledge
INT	Interrupt request
IR	I/O read
IW	I/O write
LCK	Bus priority lock control
RD	Memory reads
SS	Stack segment
TST	Test control for WAIT instruction
WT	Memory writes

CONS---Set Consecutive Events

The 808x emulators support the CYC mode of the CONS command. The 808x emulators do **not** support either the FET mode or the EMU mode of the CONS command.

DISP---Display Contents of Acquisition Memory

The DISP command displays the contents of the TTA's Acquisition Memory. The "Q" column contains the number of bytes remaining in the pre-fetch queue after each trace line. The queue may contain up to six bytes for the 8086, and up to four bytes for the 8088. Refer to the "Special Considerations" subsection for more discussion of the 8086/8088 pre-fetch queue.

Here is an example of the DISP command:

```
> DISP <CR>
ADDR  DATA  MNEM  OPER  Q  7-PROBE-0  BUS
000100 BB  MOVW  BX,#0500  1 1111 1111  F CS BHE TST LCK
000101 00                                0 1111 1111  FS CS BHE TST
000102 05                                1 1111 1111  FS CS BHE TST
000103 B9  MOVW  CX,#0005  0 1111 1111  F CS BHE TST
000104 05                                1 1111 1111  FS CS BHE TST
000105 00                                0 1111 1111  FS CS BHE TST
000106 32  XORB  AL,AL      1 1111 1111  F CS BHE TST
000107 C0                                0 1111 1111  FS CS BHE TST
000108 02  ADDB  AL,[BX]    1 1111 1111  F CS BHE TST
000109 07                                0 1111 1111  FS CS BHE TST
000500 00                                1111 1111  RD DS      TST
00010A 43  INC   BX        3 1111 1111  F CS BHE TST
00010B E2  LOOP  $-03      4 1111 1111  F CS BHE TST
00010C FB                                3 1111 1111  FS CS BHE TST
000108 02  ADDB  AL,[BX]    1 1111 1111  F CS BHE TST
000109 07                                0 1111 1111  FS CS BHE TST
000501 00                                1111 1111  RD DS BHE TST
00010A 43  INC   BX        3 1111 1111  F CS BHE TST
00010B E2  LOOP  $-03      4 1111 1111  F CS BHE TST
00010C FB                                3 1111 1111  FS CS BHE TST
000108 02  ADDB  AL,[BX]    1 1111 1111  F CS BHE TST
000109 07                                0 1111 1111  FS CS BHE TST
```

The following display contains an example of DISP output with both TRA ALL and symbolic debug on. The information at addresses 8, 9, and A appears whenever the operating system generates a break. (The system breaks at every instruction during TRA ALL.) The LCK bus transaction occasionally appears when the system resumes execution, and does not signify an error.

```
> DISP <CR>
ADDR    DATA MNEM    OPER                Q 7-PROBE-0    BUS

START
000100 BB    MOVW      BX,#0500        1 1111 1111    F  CS BHE TST LCK
000101 00                                0 1111 1111    FS CS BHE TST
000102 05                                1 1111 1111    FS CS BHE TST
000008 7F                                1111 1111    RD CS BHE TST
000009 EF                                1111 1111    RD CS BHE TST
00000A 7F                                1111 1111    RD CS BHE TST

DEMO+000103
000103 B9    MOVW      CX,#0005        0 1111 1111    F  CS BHE TST
000104 05                                1 1111 1111    FS CS BHE TST
000105 00                                0 1111 1111    FS CS BHE TST
000008 7F                                1111 1111    RD CS BHE TST
000009 77                                1111 1111    RD CS BHE TST
00000A 7F                                1111 1111    RD CS BHE TST

DEMO+000106
000106 32    XORB      AL,AL           1 1111 1111    F  CS BHE TST LCK
000107 C0                                0 1111 1111    FS CS BHE TST
000008 7F                                1111 1111    RD CS BHE TST
000009 E7                                1111 1111    RD CS BHE TST
```

SPECIAL CONSIDERATIONS

The following paragraphs describe some of the special considerations of the 808x emulators.

The HALT Instruction

In all three modes, a HALT instruction causes the emulator to halt. Control does not automatically return to the operating system. Press CTRL-C to return control to the operating system.

Losing Clock Causes Emulator Fault Error

When you select the 808x emulator, a section of system code is loaded in the emulator. This code is checked each time you start the emulator. If this code is corrupted in some way, error 45 "Emulator faulted" will occur, and you must reselect the emulator.

This error is caused by losing the clock in emulation modes 1 or 2. This may occur if you turn off your prototype while the development system is in emulation mode 1 or 2. You can avoid this error by changing to emulation mode 0 before turning off your prototype.

PROM Programmer Considerations

Due to power supply limitations, the 808x emulators must not be selected while you are programming a PROM.

Emulating the Execution Bus

Internally, the 8086 and 8088 are each dual processors. Each contains an Execution Unit (EU) and a Bus Interface Unit (BIU). The Bus Interface Unit prefetches instructions for the Execution Unit, and interfaces the EU to the outside world. The BIU communicates with the EU via an "Execution Bus".

The 808x emulators emulate the information on the Execution Bus: this means that instructions are disassembled and traced accurately. Breakpoints are generated by information on the Execution Bus.

Viewing the Pre-Fetch Queue

With the 808x emulators, you can use the Trigger Trace Analyzer's DISP command to view the depth of the pre-fetch queue. This feature tells you how many bytes remain in the pre-fetch queue after each trace line. When the queue depth is zero, the Execution Unit waits for the Bus Interface Unit to fetch the next instruction. You can optimize critical sections of code for execution speed by knowing when the queue is empty.

Segment Registers

The 8086/8088 microprocessor contains four segment registers: Code Segment (CS), Data Segment (DS), Stack Segment (SS), and Extra Segment (ES). Each segment register contains the 16 most significant bits of an 8086/8088 20-bit address. Addresses are given as a 16-bit offset to the segment base address.

The CSX, DSX, SSX and ESX Symbols. For convenience, a special symbol has been created for each of the four memory segments that is 16 times the value in its associated segment register. For example, the symbol DSX represents a value 16 times the value of the DS register.

The following example shows a use for the CSX symbol:

Example. The 8086/8087/8088 microprocessors create the effective address (EA) for a symbolic label by adding the current Code Segment (CS) times 16 to the contents of the Instruction Pointer (IP).

The 808x emulators only know the effective address (EA) of a symbol. However, the **address** parameter of the G command is assumed to contain only IP information. Under certain circumstances --- as when the EA of a symbol is greater than FFFFH, or the CS is not zero --- the 808x emulator calculates an IP that is larger than 16 bits, and a truncation error occurs.

But since you can use the EA and the CS to calculate the IP:

$$IP = EA - (16 * CS)$$

and the value of CSX is 16*CS, you can avoid truncation errors by subtracting the CSX symbol in the expression representing the desired address. For example:

```
> G START-CSX <CR>
```

The -CSX modifier subtracts the CS value from the EA, thus ensuring that the IP is no more than 16 bits.

SERVICE CALLS

Service calls (SVCs) enable your program to use many system capabilities of your 8450, 8550, or 8560.

An SVC is invoked with an 8086/8087/8088 OUT instruction followed by two NOPs. The operand of the OUT instruction is indirectly referenced through register DX whenever the SVC port range is larger than OFFH. (For example, the default SVC port range of OFFF0H--OFFF7H requires that the DX register be used.) This instruction sequence directs the system to a specified memory address called the SRB pointer (which points to an SRB -- a Service Request Block). The SRB pointer tells the system where to find the data (stored in the SRB) that tells the system which service to perform. Refer to the Service Calls section of your System Users Manual for an explanation of service calls, service request blocks, and SRB pointers.

Table 7K-5 shows the default addresses for the eight SRB pointers. These addresses and their associated port values can be altered with the SVC command to suit your program requirements. See the Command Dictionary section of your System Users Manual for syntax and use of the SVC command.

SVCs in Modes 1 and 2

The 8086/8088 emulator supports SVCs in all three emulation modes. In all three modes, use **two** NOPs following the I/O instruction. This allows time for the SVC to occur.

NOTE

In mode 1, only the instruction sequence that calls the SVC may reside in prototype memory. The SRB pointers, the SRB, and the optional I/O buffer must all reside in program memory.

SRB Format

The 8086/8088 emulator uses the LAS (Large Address Space) format for SRBs and SRB pointers. This format is described in the Service Calls section of your System Users Manual.

SVC Demonstration

Figure 7K-2 lists an 8086/8088 program that uses four SVC functions: Assign Channel, Read ASCII, Write ASCII, and Abort. The program's algorithm is explained in the Service Calls section of your System Users Manual, which demonstrates a version of the program written in 8085A assembly language. Using the program in Fig. 7K-2, you can perform a parallel demonstration with the 8086/8088 B Series Assembler and 8086/8088 emulator.

The program accepts a line of ASCII characters from the system terminal; then, when it receives a RETURN character, the program writes the line to the line printer and accepts another line. (On the 8550, output to the line printer is buffered. No text is printed until the 8550's line printer buffer becomes full or the program ends.)

Table 7K-5
808x Service Calls

SVC Number	Service Call (*a)		Default Location of SRB Pointer
	Mnemonic	Hexadecimal	
1	MOVW DX,#OFFF7H	BAF7FF	40--43
	OUT DX,AL	EE	
	NOP	90	
	NOP	90	
2	MOVW DX,#OFFF6H	BAF6FF	44--47
	OUT DX,AL	EE	
	NOP	90	
	NOP	90	
3	MOVW DX,#OFFF5H	BAF5FF	48--4B
	OUT DX,AL	EE	
	NOP	90	
	NOP	90	
4	MOVW DX,#OFFF4H	BAF4FF	4C--4F
	OUT DX,AL	EE	
	NOP	90	
	NOP	90	
5	MOVW DX,#OFFF3H	BAF3FF	50--53
	OUT DX,AL	EE	
	NOP	90	
	NOP	90	
6	MOVW DX,#OFFF2H	BAF2FF	54--57
	OUT DX,AL	EE	
	NOP	90	
	NOP	90	
7	MOVW DX,#OFFF1H	BAF1FF	58--5B
	OUT DX,AL	EE	
	NOP	90	
	NOP	90	
8	MOVW DX,#OFFF0H	BAF0FF	5C--5F
	OUT DX,AL	EE	
	NOP	90	
	NOP	90	

(*a) The default SVC port range, OFFF0--OFFF7, is assumed.

NOTE

The program shown in Fig. 7K-2 is written for a B Series assembler. To make this program acceptable for an A Series assembler (as used on some 8550s), change each single quote (') to a double quote ("). To terminate the program, enter a CTRL-Z while the program is waiting for input.

```

; SSSSS V V CCCC
; S V V C
; SSSSS V V C DEMONSTRATION. 8086/8087/8088 EMULATOR
; S V V C
; SSSSS V CCCC
    ORG 40H ; Beginning of SRB vector
    BYTE 0,BITS(SRB1FN,16,4),HI(SRB1FN),LO(SRB1FN)
    BYTE 0,BITS(SRB2FN,16,4),HI(SRB2FN),LO(SRB2FN)
    BYTE 0,BITS(SRB3FN,16,4),HI(SRB3FN),LO(SRB3FN)
    BYTE 0,BITS(SRB4FN,16,4),HI(SRB4FN),LO(SRB4FN)
    BYTE 0,BITS(SRB5FN,16,4),HI(SRB5FN),LO(SRB5FN)
; End of SRB vector
    ORG 60H ; Set up SRB areas
; SRB1 = Assign 'CONI' to Channel 0
SRB1FN BYTE 10H ; Assign
        BYTE 00H ; to Channel 0
SRB1ST BLOCK 01H ; Status returned here
        BLOCK 03H ; Bytes 4 through 6 not used
        BYTE 00H,05H ; Length of 'CONI' + <CR>
        BYTE 00H ; Pointer
        BYTE BITS(CONI,16,4) ; to
        BYTE HI(CONI) ; 'CONI'
        BYTE LO(CONI) ; + <CR>
; End of SRB1
; SRB2 = Assign 'LPT' to Channel 1
SRB2FN BYTE 10H ; Assign
        BYTE 01H ; to Channel 1
SRB2ST BLOCK 01H ; Status returned here
        BLOCK 03H ; Bytes 4 through 6 not used
        BYTE 00H,04H ; Length of 'LPT' + <CR>
        BYTE 00H ; Pointer
        BYTE BITS(LPT,16,4) ; to
        BYTE HI(LPT) ; 'LPT'
        BYTE LO(LPT) ; + <CR>
; End of SRB2
; SRB3 = Read ASCII line from CONI (Channel 0)
SRB3FN BYTE 01H ; Read ASCII
        BYTE 00H ; from Channel 0
SRB3ST BLOCK 01H ; Status returned here
        BLOCK 01H ; Byte 4 not used
        BLOCK 02H ; Byte count returned here
        BYTE 01H,00H ; 256 Bytes in our buffer
        BYTE 00H ; Pointer
        BYTE BITS(BUFFER,16,4); to
        BYTE HI(BUFFER) ; BUFFER
        BYTE LO(BUFFER) ; + <CR>
; End of SRB3

```

Fig. 7K-2. 808x SVC demonstration program listing (part 1 of 3).

```

;      SRB4 = Write ASCII line to LPT (Channel 1)
SRB4FN BYTE    02H          ; Write ASCII
      BYTE    01H          ; to Channel 1
SRB4ST BLOCK    01H          ; Status returned here
      BLOCK    01H          ; Byte 4 not used
      BLOCK    02H          ; Byte count returned here
      BYTE    01H,00H       ; 256 bytes in our buffer
      BYTE    00H          ; Pointer
      BYTE    BITS(BUFFER,16,4); to
      BYTE    HI(BUFFER)    ; BUFFER
      BYTE    LO(BUFFER)    ; + <CR>
;      End of SRB4
;
;      SRB5 = Abort (Close all channels and terminate)
SRB5FN BYTE    1FH          ; Abort
      BLOCK    0BH          ; Bytes 2 through 12 not used
;      End of SRB5
;
;
BUFFER BLOCK    100H          ; Our I/O area
CONI   ASCII    'CONI'        ; ASCII of 'CONI'
      BYTE    0DH          ; + <CR>
LPT    ASCII    'LPT'         ; ASCII of 'LPT'
      BYTE    0DH          ; + <CR>
;      End of data definitions
;
;      Beginning of executable code
START  ORG      1000H          ; Entry point into program
      MOVW     DX,#0FFF7H      ; Call SVC1
      OUT      DX,AL           ; to assign 'CONI'
      NOP                      ; to Channel 0
      NOP                      ;
      MOV      AL,SRB1ST       ; Check the status
      CMP      AL,#00H         ; to see if all went well
      JNZ      ABORT           ; No? Stop everything
      MOVW     DX,#0FFF6H      ; Call SVC2
      OUT      DX,AL           ; to assign 'LPT'
      NOP                      ; to Channel 1
      NOP                      ;
      MOV      AL,SRB2ST       ; Check the status
      CMP      AL,#00H         ; to see if all went well
      JNZ      ABORT           ; No? Stop everything

```

Fig. 7K-2. 808x SVC demonstration program listing (part 2 of 3).

```
ALoop    MOVW    DX,#0FFF5H    ; Call SVC3
          OUT     DX,AL        ; to read a line
          NOP     ; from 'CONI'
          NOP     ; into the buffer
          MOV     AL,SRB3ST    ; Check the status
          CMP     AL,#00H     ; to see if all went well
          JNZ     ABORT       ; No? Stop everything
          MOVW    DX,#0FFF4H    ; Call SVC4
          OUT     DX,AL        ; to write the line
          NOP     ; from the buffer
          NOP     ; to 'LPT'
          MOV     AL,SRB4ST    ; Check the status
          CMP     AL,00H     ; to see if all went well
          JZ      ALoop       ; Yes? Back to read another line
          ; No? Fall through to termination
ABORT     MOVW    DX,#0FFF3H    ; Call SVC5
          OUT     DX,AL        ; to exit
          NOP     ; the program
          NOP     ;
          END     START        ; End of program
```

Fig. 7K-2. 808x SVC demonstration program listing (part 3 of 3).

JUMPERS

The following paragraphs describe the jumpers on the 8086/8088 Emulator Processor boards and in the Prototype Control Probes. These jumpers enable you to configure your 808x emulator to suit your prototype application. For more information on these jumpers and their locations, refer to your 8086/8088 Emulator Processor and Prototype Control Probe Installation Service Manual.

NOTE

For all jumpers on the emulator boards, position 1 is indicated by an arrow on the circuit board, unless otherwise noted.

EMULATOR BOARD I JUMPERS

Emulator Board I has three jumpers, which are described in the following text.

P3567---SVC Mode Selector. This jumper controls the operation of service calls (SVCs) with respect to the development systems emulation mode. There are three possible positions for this jumper:

Normal (2-1) Service calls may occur only in emulation mode 0. Modes 1 and 2 must use prototype I/O.

Option (2-3) Service calls may occur in any mode, as determined by your program.

Option (2-4) Service calls may occur in modes 0 and 1. Your program determines whether system or prototype I/O is used.

P4029---Prototype Clock Match. This jumper matches the emulator clock speed to the prototype clock speed. The jumper has two positions:

Normal (2-3) HI. The prototype clock speed is from 4.5 to 8 MHz.

Option (1-2) LO. The prototype clock speed is from 2 to 4.5 MHz.

P5039---TTA Strobe Control. This jumper controls Trigger Trace Analyzer (TTA) strobing during system operations. The jumper has two positions:

Normal (2-3) TTA strobing is disabled.

Option (1-2) TTA strobing is enabled during system operations.

EMULATOR BOARD II JUMPERS

Emulator Board II has four jumpers, which are described in the following text.

P3106---Write-Protect Break Option. This jumper controls a break option upon a write to a protected area of memory. The jumper has two positions:

Normal (2-3) **YES.** The emulator breaks if your program attempts to write to a protected area of memory.

Option (1-2) **NO.** The emulator does not break on a write to protected memory.

P3505---Wait State Option Select. This jumper controls wait state generation during program memory access. There are three positions:

Normal (2-4) **DYN.** Wait states are generated only for those areas of memory that have been allocated with the AL -S option. The number of wait states generated is determined by the position of jumper P7005.

Option (1-2) **ALLS.** Wait states are generated for all memory locations. The number of wait states generated is determined by the position of jumper P7005.

Option (2-3) **ALLF.** No wait states are generated.

P5017---Break on Unavailable Memory Option. This jumper controls an option to break on an attempt to read from or write to unavailable memory. There are two positions:

Normal (1-2) **YES.** The emulator breaks when unavailable memory is accessed. An error message is displayed.

Option (2-4) **NO.** The emulator does not break when unavailable memory is accessed.

P7005---Wait State Generator. This jumper controls the number of wait states (up to seven) that are generated when a jumper is placed across pins 1-2 or pins 2-4 of jumper P3505.

P7005 consists of 14 pins. With the component side of the circuit board facing you, and the 100-tab edge connector down, P7005 forms seven two-pin columns numbered from left to right.

Each two-pin column will cause a corresponding number of wait states to be generated. For example, if a jumper is placed across column 1, one wait state will be generated. If a jumper is placed across column 6, six wait states will be generated.

PROTOTYPE CONTROL PROBE JUMPERS AND STRAPS

The following text describes the purpose and alternate positions of jumpers and straps located in the prototype control probes. With the exception of straps located on the mobile microprocessor board, the location and purpose of the jumpers in all probes is identical.

NOTE

In order to access jumpers in a prototype control probe, the housing of the probe's interface assembly must be disassembled. This procedure should be performed by a qualified service person.

Control Board Jumpers

The Control board has seven jumper positions, which are described in the following paragraphs.

P2---Outread Option. This jumper controls a path for data from program or mapped-in memory to prototype circuitry. The data path allows devices in a prototype, such as an 8087 coprocessor, to read data from program memory during emulation mode 1. The jumper has two positions:

Normal (1-2) The path of data is disabled.

Option (2-3) The path of data is enabled during emulation mode 1. You should select this option if your prototype uses a coprocessor in emulation mode 1, and the coprocessor's data is not mapped to the prototype. Note that input buffers in prototype circuitry must be disabled during outreads.

P3---DMA Disable Option. This jumper controls emulator-to-prototype DMA during system operations (when the emulator is not executing a prototype program). There are two positions:

Normal (2-3) Emulator DMA to the prototype is enabled. Select this position if DMA is continuously needed by the prototype circuit (for example, for CRT refresh).

Option (1-2) Emulator DMA to the prototype is disabled when a prototype program is not executing. Select this position if you want to use the D (Display) or EX (EXamine) commands to examine memory affected by DMA operations.

P4---Hold Timeout Option. This jumper controls a break option for DMA operations. There are two positions:

Normal (2-3) The emulator will wait indefinitely for a DMA operation to finish.

Option (1-2) If a break request occurs during a DMA operation, the emulator will break after a 0.5 second delay.

NOTE

If jumper P4 is set to optional position (1-2), jumper P3 must also be set to position (1-2).

P5---Clock Deskewer Enable. This jumper enables the emulator's clock deskewer circuitry. The jumper has two positions:

Normal (2-3) **DESKEW.** The clock deskewer is enabled.

Option (2-1) **BYPASS.** The clock deskewer is disabled. Select this position only if the prototype clock is operating at other than a steady state 33% duty cycle.

P6---Clock Deskewer Calibrator, 2 ns. This jumper is used to calibrate the clock deskewer circuit in increments of 2 ns. The time base for the deskewer will be the sum of the intervals selected by P6 and P7. Three positions are marked on the component side of the Control board:

4 An interval of 4 ns is added to the interval selected by P7.

2 An interval of 2 ns is added to the interval selected by P7.

0 Only the interval selected by P7 affects the time base.

P7---Clock Deskewer Calibrator, 6 ns. This jumper is used to calibrate the clock deskewer circuit in increments of 6 ns. The time base for the deskewer will be the sum of the intervals selected by P7 and P6. Four positions are marked on the component side of the Control board:

24 An interval of 24 ns is added to the interval selected by P6.

18 An interval of 18 ns is added to the interval selected by P6.

12 An interval of 12 ns is added to the interval selected by P6.

6 An interval of 6 ns is added to the interval selected by P6.

P8---Deskewer Prototype Clock Match. This jumper matches the prototype clock speed to the clock deskewer circuit. The jumper has two positions:

Normal (2-3) HI. The prototype clock speed is from 4.5 to 8 MHz.

Option (1-2) LO. The prototype clock speed is from 2 to 4.5 MHz.

BUFFER BOARD JUMPERS

The Buffer board has seven jumpers, which are described in the following paragraphs.

P6012---Synchronize RESET Option. This jumper selects an option that can synchronize the RESET signal from the prototype circuit with emulator bus cycles. When enabled, the option minimizes the likelihood of random writes to memory. The jumper has two positions:

Normal (1-2) The RESET synchronization option is disabled.

Option (2-3) The RESET synchronization option is enabled.

P7072---Prototype READY Control. This jumper selects an option that allows the prototype's READY signal to control access of program memory in emulation mode 1. In emulation mode 1, each program memory access causes a corresponding prototype memory access. Although data in prototype memory is not read, an access of this memory may cause the prototype to hang, if the prototype is part of a multibus system. P7072 has two positions:

Normal (2-3) The READY control option is disabled.

Option (1-2) The READY control option is enabled.

P7076---READY Fault/Timeout Wait State Select. This jumper selects the number of wait states that precede a READY fault. P7076 works in conjunction with P7077 and P7078; refer to Table 7K-6. Jumper P7076 has four positions:

Normal (7-8) Zero wait states are selected.

Option (1-2) One or two wait states are selected, depending upon prototype timing.

Option (3-4) Four or five wait states are selected, depending upon prototype timing.

Option (5-6) Eight or nine wait states are selected, depending upon prototype timing.

P7077---READY Timeout Option. This jumper controls a READY fault "timeout" option that forces READY true, after a specified number of wait states, as determined by P7076. (Refer to Table 7K-6.) The jumper has two positions:

Normal (1-2) The timeout option is disabled.

Option (2-3) The timeout option is enabled.

P7078---Break On READY Fault Option. This jumper controls a READY fault break option. When selected, the option causes a break (with an error message) following a READY fault. This jumper works in conjunction with P7076 and P7077; refer to Table 7K-6. The jumper has two positions:

Normal (2-3) The break option is disabled.

Option (1-2) The break option is enabled.

Table 7K-6
P7076, P7077 and P7078 Interactions

P7076	P7077	P7078	Function Produced
0 wait states	don't care	don't care	The emulator waits indefinitely for READY to go true.
1 or more wait states	OFF	OFF	The emulator waits indefinitely for READY to go true. However, after the specified number of wait states, any other break condition (such as CTRL-C) will force READY true and generate a break. No error message is displayed.
1 or more wait states	don't care	ON	After the specified number of wait states, the emulator times-out and forces READY true. A break is generated. An error message is displayed.
1 or more wait states	ON	OFF	After the specified number of wait states, the emulator times-out and forces READY true. No break is generated. No error message is displayed.

Dummy Address Jumpers. The Dummy Address Jumpers select a block of 256 "dummy" addresses to which pseudo-code fetches can be directed. These addresses are required because a system access to prototype memory may cause superfluous read cycles. Note that the dummy addresses must not include any memory-mapped I/O addresses or memory used for another purpose.

There are twelve dummy address jumpers (DA8---DA19). Each corresponds to one of the twelve most-significant address lines. Once a dummy address block is selected with the jumpers, any superfluous read cycles are directed to addresses in that block. The individual address that is accessed is

determined by the eight least-significant address lines (A0---A7). Table 7K-7 lists the dummy address jumpers and their corresponding address lines.

Each dummy address position has three pins. If a jumper is placed across pins 1 and 2 (default), the address line associated with that jumper will be LOW. If a jumper is placed across pins 2 and 3, the address line will be held HIGH.

Table 7K-7
Dummy Address Jumpers

Jumper	Address Line
=====	
P3051	DA8
P4051	DA9
P3041	DA10
P3053	DA11
P3042	DA12
P3052	DA13
P4031	DA14
P4032	DA15
P4073	DA16
P4074	DA17
P3076	DA18
P4076	DA19

MOBILE MICROPROCESSOR JUMPERS

Each of the four mobile microprocessor boards that support the 808x emulator has three jumpers. The purpose and positions of these jumpers are discussed in the following text.

J3011 (8086/8088) and J8013 (8087)---Microprocessor Identity Selector. This jumper identifies the emulating microprocessor. There are two positions:

- (1-2) The emulating microprocessor is identified as an 8086.
- (2-3) The emulating microprocessor is identified as an 8088.

J3013 (8086/8088) and J8014 (8087)---Clock Frequency Selector. This jumper, in conjunction with J3015 (8086/8088 emulator) or J9011 (8087 emulator), selects the clock frequency for the emulating microprocessor. There are two positions:

- (1-2) An 8 MHz clock is selected.
- (2-3) A 5 MHz clock is selected.

J3015 (8086/8088) and J9011 (8087)---Clock Frequency Selector. This jumper, in conjunction with J3013 (8086/8088 emulator) or J8014 (8087 emulator), selects the clock frequency for the emulating microprocessor. There are two positions:

(1-2) A 5 MHz clock is selected.

(2-3) An 8 MHz clock is selected.

8087 SUPPORT JUMPERS

The 8086/8087 and 8088/8087 Mobile Microprocessor boards have four jumpers that must be configured to support the 8087 coprocessor. All four jumpers must be moved together. These jumpers are described in the following text.

P7011---RQ1(L)/GT1(L) Source Selector. This jumper selects the source of the RQ1(L)/GT1(L) signal to the emulating microprocessor.

Normal (1-2) The 8087's RQ0(L)/GT0(L) signal is the source of RQ1(L)/GT1(L).

Other (2-3) The emulating microprocessor's RQ1(L)/GT1(L) signal comes from the prototype.

P7012---NMI Signal Enable. P7012 controls the path of the NMI vector address signal. This signal is decoded from the NMI vector address and presented to the READY/HOLD circuitry.

Normal (1-2) The signal is allowed to pass.

Option (2-3) The signal is disabled.

P8012---8087 RESET Selector. This jumper controls the path of a RESET signal from the prototype to the 8087 microprocessor.

Normal (1-2) The prototype's RESET signal resets the 8087 as well as the emulating microprocessor.

Option (2-3) The 8087 is held at RESET.

P8013---TEST Signal Source. This jumper controls the source of the emulating microprocessor's TEST signal.

Normal (1-2) The TEST signal comes from the 8087's BUSY signal.

Option (2-3) The TEST signal comes from the prototype's TEST signal.

POWER SUPPLY BOARD JUMPER

The Power Supply board has one jumper.

P5061---808x/68000 Power Supply Selector. This jumper configures the power supply board for the prototype control probes that support the 8086/8088 Emulator Processor module. The jumper has two positions:

Normal (1-2) This configures the power supply for the 808x Emulator Processor module. For the 808x emulators, P5061 must be set to this position.

Other (2-3) This configures the power supply for the 68000 Emulator Processor module. Note that this is not an option.

32K PROGRAM MEMORY BOARD ADJUSTMENTS

When an 808x emulator is installed, each 32K Program Memory board in the development system must be configured to support the operation of this emulator. The following text tells you how to set a strap, two jumpers, and a DIP switch on a 32K Program Memory board to provide 808x support. Refer to your development system's installation manual for information on the location and function of these jumpers and switches.

W5011---Delayed Read Strap. The ECB run between pins 1 and 2 must be cut, then a strap must be soldered between pins 2 and 3.

J6175---LOW/HIGH Board Jumper. If there is only one 32K memory board in the Program Section of your development system, place a jumper across pins 2 and 3 of J6175. If there are two 32K memory boards, place a jumper across pins 1 and 2 on one board, and across pins 2 and 3 on the other board.

J7171---Extended Bank Jumper. Place this jumper across pins 1 and 2.

S7170---Extended Memory DIP Switch. Set all of the switches on this DIP switch to 0 (the ON position).

64K SYSTEM MEMORY BOARD ADJUSTMENTS

When an 808x emulator is installed, each 64K Program Memory board in the development system must be configured to support the operation of this emulator. The following text tells you how to set two jumpers on a 64K memory board to provide 808x support. Refer to your development system's installation manual for information on the location and function of these jumpers.

J7090---Read Delay Jumper. Set this jumper to the 0 ns position (no delay).

J8100--J8160---Address Select Jumpers. Set these jumpers to select continuous memory from 0 to 128K. To do this, set jumper group A to 0, group B to 32K, group C to 64K, and group D to 96K. Refer to the 64K/128K Static Program Memory Installation Manual for more information on how to set these jumpers.

EMULATOR TIMING

The signals between the prototype and the emulating microprocessor are buffered. Therefore, some timing differences exist between the 8086/8088 emulator and an 8086 or 8088 microprocessor inserted directly into the prototype.

Table 7K-8 lists the emulator/microprocessor timing differences for the 8086/8088 MX (Maximum) Mode (worst case conditions). Figure 7K-3 contains timing diagrams corresponding to the signals listed in Table 7K-8.

NOTE

The numbers in the left most column of Tables 7K-8 and 7K-9 are used to identify the signals on the timing diagrams. Some of the signals in Table 7K-8 also appear in Table 7K-9 and are given the same numbers in both tables. In Table 7K-9, the signals are arranged in the order in which they appear in the Intel 8086 specifications.

Timing information is from **The 8086 Family User's Manual** published by Intel Corporation, October, 1979. Used with permission.

Table 7K-8
Representative 808x Emulator/Microprocessor Timing Differences (MX Mode)

	Symbol	Parameter	Processor		Emulator		Units
			Min.	Max.	Min.	Max.	
1	TCLCL	CLK cycle period	125	500	125	500	ns
2	TCLCH	CLK low time (*a)	$(2/3 \text{ TCLCL}) - 15$		$(2/3 \text{ TCLCL}) - 15$		ns
3	TCHCL	CLK high time (*a)	$(1/3 \text{ TCLCL}) - 15$		$(1/3 \text{ TCLCL}) - 15$		ns
4	TCH1CH2	CLK rise time		10		10	ns
5	TCL2CL1	CLK fall time		10		10	ns
6	TDVCL	Data in setup time	20		31		ns
7	TCLDX	Data in hold time	10		3		ns
8	TRYHCH	READY setup time into 8086 (*a)	$(2/3 \text{ TCLCL}) - 15$		$(2/3 \text{ TCLCL}) + 2$		ns
9	TCHRYX	READY hold time into 8086	20		3		ns
10	TRYLCL	READY inactive to CLK (*b)	-8		6		ns

Table 7K-8
(Cont)

	Symbol	Parameter	Processor		Emulator		Units
			Min.	Max.	Min.	Max.	
11	TINVCH	Setup time for recognition (INTR, NMI, TEST(L)) (*c)	15		15		ns
12	TGVCH	RQ(L)/GT(L) setup time	15		15		ns
13	TCHGX	RQ(L) hold time into 8086	30		30		ns
14	TRYHSH	READY active to status passive (*d)		65		82	ns
15	TCHSV	Status active delay	10	60	15	71	ns
16	TCLSH	Status inactive delay	10	70	15	81	ns
17	TCLAV	Address valid delay	10	60	15	71	ns
18	TCLAX	Address hold time	10		15		ns
19	TCLAZ	Address float delay	TCLAX	50	TCLAX	71	ns
20	TCLDV	Data valid delay	10	60	15	71	ns
21	TCHDX	Data hold time	10		3		ns
22	TAZRL	Address float to read active	0		0		ns
23	TCLRL	RD active delay	10	100	19	60	ns
24	TCLRH	RD inactive delay	10	80	18	57	ns
25	TRHAV	RD inactive to next address active	TCLCL-40		TCLCL-35		ns
26	TCLGL	GT(L) active delay	0	50	0	50	ns
27	TCLGH	GT(L) inactive delay	0	50	0	50	ns
28	TRLRH	RD(L) width (*a)	2TCLCL-50		2TCLCL-75		ns

(*a) Minimum value only.

(*b) Applies only to T2 state (8 ns into T3).

(*c) Setup requirement for asynchronous signal only to guarantee recognition at next CLK.

(*d) Applies only to T3 and wait states.



Timing diagram courtesy of Intel Corporation, Copyright 1979. Used with permission.

Table 7K-9 lists the emulator/microprocessor timing differences for the 808x microprocessor in MN (Minimum) Mode (worst case conditions). Figure 7K-4 contains a timing diagram corresponding to the signals listed in Table 7K-9.

Timing information is from **The 8086 Family User's Manual** published by Intel Corporation, October, 1979. Used with permission.

Table 7K-9
Representative 808x Emulator/Microprocessor Timing Differences (MN Mode)

	Symbol	Parameter	Processor Min. Max.	Emulator Min. Max.	Units
1	TCLCL	CLK cycle period	125 500	125 500	ns
2	TCLCH	CLK low time (*a)	(2/3 TCLCL)-15	(2/3 TCLCL)-15	ns
3	TCHCL	CLK high time (*a)	(1/3 TCLCL)+2	(1/3 TCLCL)+2	ns
4	TCH1CH2	CLK rise time	10	10	ns
5	TCL2CL1	CLK fall time	10	10	ns
6	TDVCL	Data in setup time	20	31	ns
7	TCLDX	Data in hold time	10	3	ns
8	TRYHCH	READY setup time into 8086 (*a)	(2/3 TCLCL)-15	(2/3 TCLCL)+2	ns
9	TCHRYX	READY hold time into 8086	20	3	ns
10	TRYLCL	READY inactive to CLK (*b)	-8	6	ns
29	THVCH	HOLD setup time	20	20	ns
11	TINVCH	Setup time for recognition (INTR, NMI, TEST(L)) (*c)	15	15	ns
17	TCLAV	Address valid delay	10 60	15 71	ns
18	TCLAX	Address hold time	10	15	ns
19	TCLAZ	Address float delay	TCLAX 50	TCLAX 71	ns
30	TLHLL	ALE width	TCLCH-10	TCLCH-10	ns
31	TCLLH	ALE active delay	50	32	ns
32	TCHLL	ALE inactive delay	55	35	ns

Table 7K-9
(Cont)

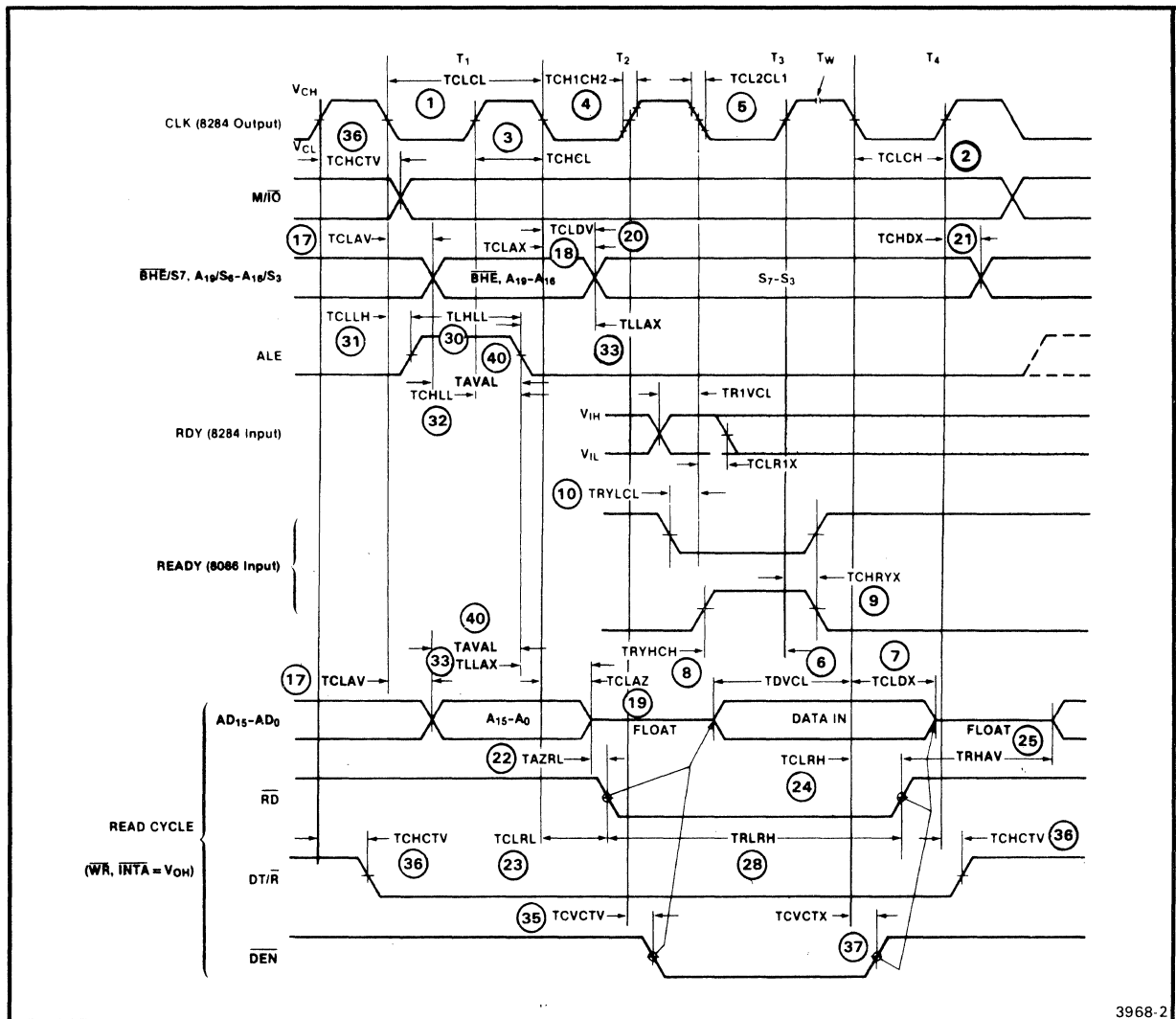
	Symbol	Parameter	Processor		Emulator		Units
			Min.	Max.	Min.	Max.	
33	TLLAX	Address hold time to ALE inactive	TCHCL-10		TCHCL-10		ns
20	TCLDV	Data valid delay	10	60	15	71	ns
21	TCHDX	Data hold time	10		3		ns
34	TWHDX	Data hold time after WR	TCLCH-30		TCLCH-30		ns
35	TCVCTV	Control active delay 1	10	70	13	65	ns
36	TCHCTV	Control active delay 2 with M/IO(L)	10	60	17	80	ns
					TCHCL+17 TCHCL+65		
37	TCVCTX	Control inactive delay	10	70	17	62	ns
22	TAZRL	Address float to READ active	0		0		ns
23	TCLRL	RD(L) active delay	10	100	19	60	ns
24	TCLRH	RD(L) inactive delay	10	80	18	57	ns
25	TRHAV	RD(L) inactive to next address active	TCLCL-40		TCLCL-35		ns
38	TCLHAV	HLDA valid delay	10	100	10	100	ns
28	TRLRH	RD(L) width (*a)	2TCLCL-50		2TCLCL-75		ns
39	TWLWH	WR(L) width (*a)	2TCLCL-40		2TCLCL-40		ns
40	TAVAL	Address valid to ALE low	TCLCH-40		TCLCH-51		ns

(*a) Minimum value only.

(*b) Applies only to T2 state (8 ns into T3).

(*c) Setup requirement for asynchronous signal only to guarantee recognition at next CLK.

(*d) Applies only to T3 and wait states.



3968-2

Fig. 7K-4. 8086 MN Mode timing diagrams.

Timing diagram courtesy of Intel Corporation, Copyright 1979. Used with permission.

INSTALLING YOUR 808X EMULATOR SOFTWARE**8540 FIRMWARE INSTALLATION PROCEDURE**

The ROMs that contain the control firmware for your 808x emulator must be installed in your 8540's System ROM board. Refer to your 8540 Emulator Installation Manual for instructions on installing these ROMs.

8550 EMULATOR SOFTWARE INSTALLATION PROCEDURE

Your emulator software package consists of two disks:

- a disk containing emulator control software, which you install onto your DOS/50 system disk so that DOS/50 can control your emulator hardware.
- a disk containing 808x diagnostic software.

This subsection describes how to install the control software for your 808x emulator.

To complete the emulator control software installation procedure you need the following items:

- an 8550 system (with or without an 808x emulator)
- a DOS/50 Version 2.1 system disk with a write-enable tab over the write-protect slot
- an 808x emulator software installation disk with no write-enable tab

This installation procedure takes about five minutes.

Start Up and Set the Date

Turn on your 8550 system. (For start-up instructions, refer to the Learning Guide of your System Users Manual.) Place your system disk in drive 0 and shut the drive 0 door. When you see the > prompt on your system terminal, place your installation disk in drive 1 and shut the drive 1 door.

Use the DAT command to set the date and time. For example, if it is 11:05 AM on October 28, 1983, type:

```
> DAT 28-OCT-83/11:05 <CR>
```

The system will use this information when it sets the CREATION time attribute of each file copied from your installation disk.

Install the Emulator Control Software

The command file INSTALL2, which installs the emulator control software, resides on the installation disk.

To execute the command file, simply type its filespec:

```
> /VOL/EMU.8086/INSTALL2 <CR>
```

DOS/50 responds with the following message:

```
* During this installation procedure, one or more of the
* following messages may appear.  IGNORE THESE MESSAGES:
*
*      Error 6E - Directory alteration invalid
*      Error 7E - Error in command execution
*      Error 1D - File not found
*
* If any OTHER error message appears, see your
* Users Manual for further instructions.
*
* If no other error message appears, you'll receive a
* message when the installation procedure is complete.
*
T,OFF
```

In this installation procedure, you may disregard error messages 6E, 7E, and 1D; these messages have no bearing on the success of the installation. However, if a message other than 6E, 7E, or 1D appears, take the following steps:

1. Make sure you are using the right disks.
2. Make sure your system disk has a write-enable tab.
3. Make sure there are at least three free files and 20 free blocks on your system disk.
4. Begin the installation procedure again.

If the installation procedure fails again, copy down the error message and contact your Tektronix service representative.

The "T,OFF" command suppresses subsequent output to your system terminal (except error messages) until INSTALL2 finishes executing. Within about five minutes, INSTALL2 will finish and your system terminal will display the following message:

```
*
* Your installation has been successfully completed.
>
```

Once your software is installed, you can:

- remove your disks and turn off your 8550 system, or
- install more software onto your system disk, or
- continue with the 8086 Emulator Demonstration Run that follows in this section. If you do this, you do not have to restart the system or reset the date and time.

NOTE

At this point, "NO.NAME" is the current user. To change the current user back to "yourname", enter **USER,,yourname**.

8086 DEMONSTRATION RUN**INTRODUCTION**

This demonstration run shows you how to load, execute, and monitor a simple 8086 assembly language program on your 8540 or 8550. In order to perform this demonstration, your 808x emulator hardware and control software must be installed in your 8540 or 8550.

Figure 7K-5 shows the source and object code for the demonstration program.

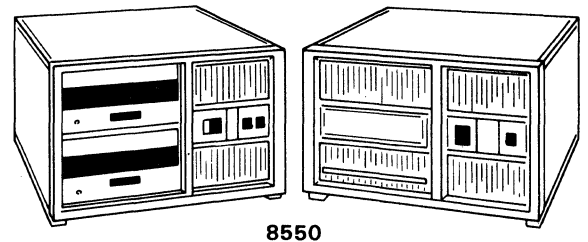
Case 1: If you have an 8550, the source code and object code for the demonstration program are provided on the installation disk that contains your 808x emulator control software. This demonstration shows you how to assemble the program on your 8550. (If your system disk does not contain an 8086/8088 assembler, you will have to skip that part of the demonstration.)

Case 2: If you have an 8540/8560 system, and your 8560 has an 8086/8088 assembler installed, you can create and assemble the program on the 8560 and then download it to the 8540. This demonstration shows how.

Case 3: If you have an 8540 that is connected to a host computer other than an 8560, we can't give you a specific list of commands for creating and assembling the program on your host (since we don't know what host you're using). However, Fig. 7K-6 gives the object code for the program in Extended Tekhex format. You can create the Tekhex file using your host's assembler or text editor, then download the file to the 8540 via the 8540's optional COM interface.

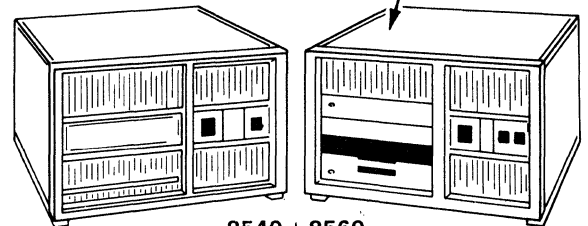
Case 4: If none of these cases applies to you, you can patch the program into memory using the P command. This demonstration shows how.

Case 1:



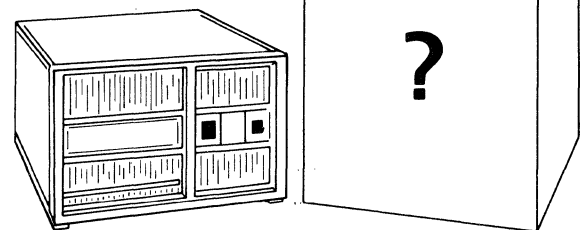
8550

Case 2:



8540 + 8560

Case 3:



8540 + other host

Case 4:

any other configuration

(3964-5)3968-3

Once the program is loaded or patched into memory, you can execute the program on your emulator.

01				;8086 DEMONSTRATION RUN PROGRAM	
02				SECTION DEMO	
03			ORG 100H		;START PROGRAM CODE AT ADDRESS 100
04	00000100	BB0005	START	MOVW BX,#TABLE	;SET TABLE POINTER
05	00000103	B90500		MOVW CX,#TSIZE	;SET PASS COUNTER
06	00000106	32C0		XORB AL,AL	;CLEAR ACCUMULATOR
07	00000108	0207	ALoop	ADDB AL,[BX]	;ADD BYTE FROM TABLE AND
08	0000010A	43		INCW BX	; POINT TO NEXT BYTE
09	0000010B	E2FB		LOOP ALoop	;DECREMENT CX AND LOOP UNTIL CX=0
10	0000010D	BAF7FF		MOVW DX,#0FFF7H	;I/O ADDRESS FOR EXIT SVC
11	00000110	EE		OUT DX,AL	;CALL EXIT SVC
12	00000111	90		NOP	; TO END PROGRAM
13	00000112	90		NOP	; EXECUTION
14				;SRB POINTER	
15				ORG 40H	;STORE SRB POINTER AT ADDRESS 40
16	00000040	00000044		BYTE 0H,0H,0H,44H	;POINT TO SRB FOR EXIT SVC
17				;SRB FOR EXIT SVC	
18	00000044	1A		BYTE 1AH	;1AH = FUNCTION CODE FOR EXIT SVC
19				;TABLE OF NUMBERS TO BE ADDED	
20				TSIZE EQU 5	;TABLE SIZE = 5
21				ORG 500H	;SET UP TABLE AT ADDRESS 500
22			TABLE	BLOCK TSIZE	
23				LIST DBG	
24				END START	

(A)

```
%2F6113100BB0005B9050032C0020743E2FBBAF7FFEE9090
%12623240000000441A
%3B3574DEMO010350515AL00P310815START310015TABLE350025TSIZE15
%098153100
```

(B)

FIRST DATA BLOCK: object code for addresses 100--112

```

header
|
| load address      object code
|
|-----|
%2F6113100BB0005B9050032C0020743E2FBBAF7FFEE9090

```

SECOND DATA BLOCK: object code for addresses 40--44

header load object
address code

%12623240000000441A

SYMBOL BLOCK

```

header      section
|           |
section     definition
name        field
|           |
symbol      definition fields
|           |
%3B3574DEMO010350515ALOO P310815START310015TABLE350025TSIZE15

```

TERMINATION BLOCK

```

header
|
transfer
address
|
%098153100

```

3968-6

Fig. 7K-6. 8086 demonstration run program: Extended Tekhex format.

EXAMINE THE DEMONSTRATION PROGRAM

The demonstration program adds five numbers from a table stored in locations 500--504 in program memory and leaves the sum in register AL. (You will place values in the table later in this demonstration.) The 8085A emulator demonstration run in the Learning Guide of your System Users Manual contains a flowchart that illustrates the steps of the program.

The source code contains two kinds of statements: assembler directives (like `ORG` and `BYTE`) and 8086 assembly language instructions. The assembler directives are microprocessor-independent and are explained in the 8085A emulator demonstration run. The 8086 assembly language instructions are discussed in the following paragraphs.

Set Table Pointer. The `MOVW BX,#TABLE` instruction loads the address of the table (500) into register BX. As a result, BX points to the first element of the table. The label `START` is used by the `END` directive to specify that the `MOVW BX,#TABLE` instruction is the first to be executed.

Set Pass Counter. Register CX is used as the pass counter. The `MOVW CX,#TSIZE` instruction moves the value of `TSIZE` into CX. This step sets the number of passes to 5.

Clear Accumulator. The `XORB AL,AL` instruction zeros register AL, so that you may use AL as the accumulator and start adding numbers from the table into it.

Add Byte from Table. The `ADDB AL,[BX]` instruction adds the byte addressed by BX into the accumulator AL. The label `ALOOP` represents the address of this instruction; this label is used by the `LOOP` instruction.

Point to Next Byte. The `INCW BX` instruction increments register BX, which then points to the next byte in the table. For example, BX is initialized to contain 500H. After the `INCW BX` instruction is first executed, BX will contain 501, the address of the second element of the table.

Decrement CX and Loop Until CX = Zero. The `LOOP ALOOP` instruction decrements register CX, the pass counter. If CX is not yet zero, the program jumps back to the label `ALOOP`. If CX is equal to zero, the program proceeds to call the `Exit SVC` routine.

Call Exit SVC. The last four instructions of the program are:

```
MOVW DX,#0FFF7H
OUT DX,AL
NOP
NOP
```

These instructions constitute a service call (SVC) that causes an exit from the program. For more information on SVCs, refer to the Service Calls discussion elsewhere in this Emulator Specifics section.

ASSEMBLE AND LOAD THE DEMONSTRATION PROGRAM

Now it's time to create the program so you can run it on your emulator. One of the following discussions describes the set of steps that is appropriate for your hardware configuration:

- **For 8550 users:** Case 1: Assemble and Load on the 8550
- **For 8540/8560 users:** Case 2: Assemble on the 8560; Download to the 8540
- **For 8540 users with host computers other than the 8560:** Case 3: Download from Your Host to the 8540
- **For other hardware configurations:** Case 4: Patch the Program into Memory.

Go ahead and work through the discussion that's appropriate for you. Once you've put the program into program memory, turn to the heading "Run the Demonstration Program," later in this section.

CASE 1: ASSEMBLE AND LOAD ON THE 8550

This discussion shows you how to copy the demonstration program from your 8086/8087/8088 emulator software installation disk, assemble the program, and load it into 8550 program memory.

Start Up and Log On

Turn on your 8550 system. (For start-up instructions, refer to the paragraph "Start Up the 8550 and Its Peripherals" in the Learning Guide of your System Users Manual.) Place your system disk in drive 0 and shut the drive 0 door. When your system displays the > prompt, place your 8086/8087/8088 emulator software installation disk in drive 1 and shut the drive 1 door.

Use the DAT command to set the current date and time. For example, if it is 2:30 PM on October 31, 1983, enter the following command line:

```
> DAT 31-OCT-83/2:30 PM <CR>
```

Use the SEL command to tell DOS/50 to use the assembler and emulator software designed for the 8086/8087/8088 microprocessors.

```
> SEL 8086 <CR>
```

The system responds with the current version number:

```
8086 emulator V n.nn mm/dd/yy
```

The SEL command automatically sets the emulation mode to 0.

Copy the Demonstration Run Program from the Installation Disk

Enter the following command lines to create an empty directory called DEMO on your system disk and make DEMO the current directory. The BR command creates a brief name, ROOT, to mark the old current directory. At the end of this demonstration, you will return to this ROOT directory and delete the DEMO directory and its contents.

```
> BR ROOT /USR <CR>
```

```
> CREATE DEMO <CR>
```

```
> USER DEMO <CR>
```

Now use the COP command to copy all the files in the DEMO2 directory on the installation disk to the DEMO directory you just created:

```
> COP /VOL/EMU.8086/DEMO2/* * <CR>
```

Remove your installation disk from drive 1 and put the disk away.

Now list the files you have just copied to the current directory:

```
> L <CR>
FILENAME
```

```
ASM
LOAD
```

```
Files used      124
Free files      132
Free blocks     821
Bad blocks      0
```

The file named ASM contains the assembly language source code for this demonstration program, and the file named LOAD contains the executable object code. This copy of LOAD will be used in the demonstration only if you do not have an 8086/8088 assembler and so cannot create your own object file and load file from the source file.

Examine the Demonstration Program

Enter the following command line to display the source file ASM on the system terminal: (Your display may differ slightly from this due to the tab settings on your terminal.)

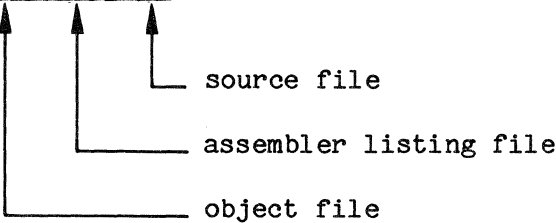
```
> CON ASM <CR>
;8086 DEMONSTRATION RUN PROGRAM
SECTION DEMO
ORG 100H ; START PROGRAM CODE AT ADDRESS 100 HEX
START MOVW BX,#TABLE ; SET TABLE POINTER
      MOVW CX,#TSIZE ; SET PASS COUNTER
      XORB AL,AL ; CLEAR ACCUMULATOR
ALOOP ADDB AL,[BX] ; ADD BYTE FROM TABLE
      INCW BX ; POINT TO NEXT BYTE
      LOOP ALOOP ; DECREMENTS CX AND LOOPS UNTIL CX=0
      MOVW DX,#0FFF7H ; I/O ADDRESS FOR EXIT SVC
      OUT DX,AL ; CALL EXIT SVC
      NOP ; TO END PROGRAM
      NOP ; EXECUTION
;SRB POINTER
ORG 40H ; STORE SRB POINTER AT ADDRESS 40 HEX
      BYTE 0H,0H,0H,44H ;POINT TO SRB FOR EXIT SVC
;SRB FOR EXIT SVC
      BYTE 1AH ; 1AH = FUNCTION CODE FOR EXIT SVC
;TABLE OF NUMBERS TO BE ADDED
TSIZE EQU 5 ; TABLE SIZE = 5
      ORG 500H ; SET UP TABLE AT ADDRESS 500 HEX
TABLE BLOCK TSIZE
      LIST DBG
      END START
```

Assemble the Source Code

If you do not have an 8086/8088 assembler on your system disk, you cannot perform this step, so skip the next four commands (ASM, COP, LINK, and L).

The ASM (assemble) command translates assembly language (source code) into binary machine language (object code). The ASM command also creates an assembler listing that can be used to correlate the object code with the source code. Enter the following command line to assemble the source code in the file ASM and create the listing and object files ASML and OBJ:

```
> ASM OBJ ASML ASM <CR>
```



```
Tektronix      8086 Vxx.xx-xx Copyright (c) 1981 Tektronix
**** Pass 2
  24 Source Lines  24 Assembled Lines  xxxxx Bytes Available
>>> No assembly errors detected <<<
```

Enter the following command to copy the assembler listing to the line printer. (First, make sure the printer is turned on and properly connected.)

```
> COP ASML LPT <CR>
```

The different fields of your source listing are shown in Fig. 7K-5, earlier in this demonstration. For a detailed explanation of assembler listings, consult your Assembler Core Users Manual.

Link the Object Code. The linker creates an executable load file from one or more object files. Enter the following linker command to create a load file called LOAD from your object file, OBJ:

```
> LINK -O OBJ -o LOAD -d <CR>
```

The linker command options **-O** and **-o** specify the object file and load file, respectively. The **-d** command option causes the linker to pass the program symbols from the object file along to the load file, for use in program debugging.

The files generated by the ASM and LINK commands should now be on your disk. Enter the following command to list the files in your current directory:

```
> L <CR>
FILENAME
```

```
ASM
LOAD
OBJ
ASML
```

```
Files used      126
Free files      130
Free blocks     811
Bad blocks      0
```

Notice that there are now four files listed in your directory. OBJ and ASML were created by the assembler, and LOAD was created by the linker.

Load the Program into Memory

Now it's time to load the object code from the load file LOAD into program memory.

Allocate Memory. Enter the following command to allocate a 4K-byte block of program memory to logical addresses 00000--00FFFH.

```
> AL 0 <CR>
```

Zero Out Memory. Before you load any code, use the F (Fill) command to fill program memory with zeros. Later, when you examine memory, the zeros make it easy to identify the beginning and end of your code. (Zeroing out memory has no effect on how the program is loaded.) Enter the following command line to fill memory addresses 40--11F with zeros:

```
> F 40 11F 00 <CR>
```

Check That Memory Was Filled with Zeros. Check the contents of memory with the D (Display) command. The display shows the data in hexadecimal format, and also shows the corresponding ASCII characters. Display the contents of memory addresses 40--11F with the following command line:

Load the Object Code into Memory. Enter the following command line to load the object code for the demonstration program into program memory:

load file

```
> SYMLO -S <LOAD <CR>
```

You've assembled and linked the demonstration program and loaded it into memory. Now skip ahead to the heading "Run the Demonstration Program."

CASE 2: ASSEMBLE ON THE 8560; DOWNLOAD TO THE 8540

This discussion shows you how to create the demonstration program source code and assemble it on the 8560, then download it to 8540 (or 8550) program memory. If your 8560 does not have an 8086/8088 assembler, you cannot complete this part of the demonstration, so skip ahead to the heading "Case 4: Patch the Program into Memory" for instructions.

Start Up and Log In

Start up your 8540, make sure it's in TERM mode, and log in to the 8560 operating system, TNIX. See your 8560 System Users Manual for details.

Since you're logged in to TNIX, your system prompt is "\$". (Later in the demonstration, we'll show the system prompt as ">", in deference to people using 8540s and 8550s in LOCAL mode.) Every command you enter is processed by TNIX. If you enter an OS/40 command, TNIX passes it to the 8540.

Enter the following commands to select the 8086/8088 assembler on the 8560 and an 808x emulator on the 8540:

```
$ uP=8086; export uP <CR>
$ sel 8086 <CR>
```

The **sel** command automatically sets the emulation mode to 0.

Create the Demonstration Program

Enter the following TNIX command lines to create an empty directory called demo and make demo the working directory. You'll create your source file and related files in this demo directory.

```
$ mkdir demo <CR>
$ cd demo <CR>
```

Now use the TNIX editor, **ed**, to create the demonstration program source file. The following command line invokes the editor and specifies that you want to create a file called **asm**:

```
$ ed asm <CR>
?asm
```

The editor responds "?asm" to remind you that asm does not already exist. Notice that the editor does **not** give a prompt to let you know it's ready for input.

Enter the Text. Now enter the editor command **a** (append text) and type in the program. Use the BACKSPACE key to erase typing mistakes.

```

a <CR>
      column      column      column
      8           16          27
      ↓           ↓           ↓
;8086 DEMONSTRATION RUN PROGRAM <CR>
SECTION DEMO <CR>
ORG      100H      ;START PROGRAM CODE AT ADDRESS 100 <CR>
START MOVW BX,#TABLE ;SET TABLE POINTER <CR>
MOVW CX,#TSIZE ;SET PASS COUNTER <CR>
XORB AL,AL ;CLEAR ACCUMULATOR <CR>
ALOOP ADDB AL,[BX] ;ADD BYTE FROM TABLE AND <CR>
INCW BX ; POINT TO NEXT BYTE <CR>
LOOP ALOOP ;DECREMENTS CX AND LOOPS UNTIL CX=0 <CR>
MOVW DX,#0FFF7H ;I/O ADDRESS FOR EXIT SVC <CR>
OUT DX,AL ;CALL EXIT SVC <CR>
NOP ; TO END PROGRAM <CR>
NOP ; EXECUTION <CR>
;SRB POINTER <CR>
ORG 40H ;STORE SRB POINTER AT ADDRESS 40 <CR>
BYTE 0H,0H,0H,44H ;POINT TO SRB FOR EXIT SVC <CR>
;SRB FOR EXIT SVC <CR>
BYTE 1AH ;1AH = FUNCTION CODE FOR EXIT SVC <CR>
;TABLE OF NUMBERS TO BE ADDED <CR>
TSIZE EQU 5 ;TABLE SIZE = 5 <CR>
ORG 500H ;SET UP TABLE AT ADDRESS 500 <CR>
TABLE BLOCK TSIZE <CR>
LIST DBG <CR>
END START <CR>
. <CR>

```

At the end of your text, enter a period on a line by itself. The editor will go back to accepting commands.

Check for Errors. Enter the following editor command to display the text you have entered. Check for typing mistakes.

```

1,$p <CR>
├── print command: displays the lines
│   in the designated range
├── designates last line in file
└── designates first line in file

```

If you made any mistakes, go ahead and fix them. In case you're not familiar with **ed**, Table 7K-10 lists the commands you need in order to add, delete, or replace any line. For more information on **ed**, refer to your 8560 System Users Manual.

Table 7K-10
Basic 8560 Editing Commands

Command	Function
mm,nnp <CR>	Displays lines mm through nn
nn <CR>	Makes line nn the current line
d <CR>	Deletes the current line
a <CR> <line(s) of text> . <CR>	Adds text after the current line
c <CR> <line(s) of text> . <CR>	Replaces the current line with the text you type in

Once your text is correct, enter the **w** command to write the text to the source file, asm:

```
w <CR>
902
```

The editor responds with the number of characters it wrote to the file.

Finally, enter the **q** command to quit the editor and return to TNIX:

```
q <CR>
$ ← TNIX prompt
```

Assemble the Source Code.

The TNIX **asm** (assemble) command translates assembly language (source code) into binary machine language (object code). The **asm** command also creates an assembler listing which can be used to correlate the object code with the source code. Enter the following command line to assemble the source code in the file asm and create the listing and object files asml and obj:

```
$ asm obj asml asm <CR>
```

source file

assembler listing file

object file

```
Tektronix ASM 8086
Vxx.xx-xx (8560)
*****Pass 2
```

```
24 Lines Read
24 Lines Processed
0 Errors
```

Enter the following command to print the assembler listing on the 8560's line printer:

```
$ lp1r asml <CR>
```

Check page 1 of your listing. Did the assembler issue any error messages? There should be none. If your source code contains errors, take the following steps:

1. Refer to your Assembler Users Manual to find out what the error messages mean.
2. Enter the command `ed asm` to get back into the editor and fix the mistakes in your source code. Exit the editor with the `w` and `q` commands, as before.
3. Enter the command `asm obj asml asm` to re-assemble your source code.

Link the Object Code

The linker creates an executable load file from one or more object files. Enter the following command to create a load file called load from your object file, obj. Be sure to capitalize all parameters as shown.

```
$ link -d -O obj -o load <CR>
```

The `-d` option causes the linker to pass the program symbols from the object file to the load file, for use in programming debugging.

The files generated by the `asm` and `link` commands should now be in your working directory, demo. Enter the following command to list the files in your working directory:

```
$ ls <CR>
asm
asml
load
obj
```

Notice that there are now four files listed in your directory: obj and asml were created by the assembler, and load was created by the linker.

Download the Program to the 8540

Now it's time to download the object code produced by the 8560's linker into 8540 program memory.

Allocate Memory. Enter the following command to allocate a 4K-byte block of program memory to logical addresses 00000--00FFFH.

```
$ a1 0 <CR>
```

Zero Out Memory. Before you download any code, use the OS/40 F (Fill) command to fill 8540 program memory with zeros. Later, when you examine memory, the zeros make it easy to identify the beginning and end of your code. (Zeroing out memory has no effect on how the program is loaded.) Enter the following command line to fill memory addresses 40--11F with zeros:

```
$ f 40 11f 00 <CR>
```

Check That Memory Was Filled with Zeros. Check the contents of memory with the OS/40 D (Display) command. The display shows the data in hexadecimal format, and also shows the corresponding ASCII characters. Display the contents of memory addresses 40--11F with the following command line:

```
$ d 40 11f <CR>
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
000040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
```

Download the Object Code. Enter the following command line to download the object code from the 8560 file load to 8540 program memory:

```
$ lo <load <CR>
```

load file

Download the Program Symbols. Recall that the source code for the demonstration program contains the directive LIST DBG. Because of this directive, the object file contains a list of the symbols that appear in the source code, and the values associated with those symbols. Because you included the -d option in the link command line, those symbols were passed to the load file. Use the OS/400 SYMLO command to download those symbols into the symbol table in 8540 system memory.

```
$ symlo -s <load <CR>
```

The **-s** option means that both addresses and scalars are downloaded. If you omit the **-s** option, only addresses are downloaded. (A scalar is a number that is not an address --- for example, TSIZE, the length of the table.)

Later in this demonstration, whenever you use a symbol in an OS/400 command line, OS/400 refers to the symbol table to find the value that the symbol stands for.

You've assembled and linked the demonstration program and downloaded it into memory. Now skip ahead to the heading "Run the Demonstration Program."

CASE 3: DOWNLOAD FROM YOUR HOST TO THE 8540

This discussion gives some general instructions for downloading the demonstration program from an unspecified host computer to 8540 (or 8550) program memory. If your 8540 is not equipped with the optional COM Interface Package, you cannot complete this part of the demonstration, so skip ahead to the heading "Case 4: Patch the Program into Memory" for instructions. COM Interface software is standard on the 8550.

Since we don't know what host computer you're using, we can only provide a general outline for creating the demonstration program and downloading it to the 8540. Once you have determined the command sequence that is appropriate for your host, record this information in the space provided in Fig. 7K-7.

Create the Extended Tekhex Load Module**Prepare the 8540**

(Start up the 8540.)

```
> SEL 8086 <CR>  
> AL 0 <CR>  
> F 40 11F 00 <CR>  
> D 40 11F <CR>
```

Establish Communication**Download the Load Module****Terminate Communication**

Fig. 7K-7. Host computer commands for preparing demonstration program.

Create the Extended Tekhex Load Module

In order for the object code to be downloaded to the 8540, it must be in Extended Tekhex format, as shown in Fig. 7K-6, earlier in this demonstration. You can create the load module in one of two ways:

1. Use your host computer's text editor, and type in the load module.
2. Use your host computer's 8086/8087/8088 assembler:
 - a. Translate the demonstration program into the language of your host's 8086/8087/8088 assembler.
 - b. Create and assemble the source file.
 - c. Link the object code, if necessary.
 - d. Translate the object code produced by the assembler or linker into Extended Tekhex format. The Intersystem Communication section of your System Users Manual provides a general algorithm for conversion to Extended Tekhex format.

Prepare the 8540

Start up your 8540 and enter the following command to select the 808x emulator:

```
> SEL 8086 <CR>
```

The SEL command automatically sets the emulation mode to 0, and displays the current version number.

Allocate Memory. Enter the following command to allocate a 4K-byte block of program memory to logical addresses 00000--00FFFH.

```
> AL 0 <CR>
```

Zero Out Memory. Before you download any code, use the OS/40 F (Fill) command to fill 8540 program memory with zeros. Later, when you examine memory, the zeros make it easy to identify the beginning and end of your code. (Zeroing out memory has no effect on how the program is loaded.) Enter the following command line to fill memory addresses 40--11F with zeros:

```
> F 40 11F 00 <CR>
```

Check That Memory Was Filled with Zeros. Check the contents of memory with the OS/40 D (Display) command. The display shows the data in hexadecimal format, and also shows the corresponding ASCII characters. Display the contents of memory addresses 40--11F with the following command line:

```
> D 40 11F <CR>
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
000040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
```

Download the Load Module to the 8540

Be sure that your 8540 and your host computer are connected via an RS-232-C-compatible communications link. Then perform the following steps to download the Tekhex load module to 8540 program memory. (Refer to the Intersystem Communication section of your System Users Manual to determine the commands and parameters that are appropriate for your host computer.)

- a. Enter the 8540 COM command to establish communication. (The parameters of the COM command are host-specific.) Log on to your host and execute any necessary host initialization commands.
- b. Enter the command line that downloads the Tekhex load module to the 8540. This command line consists of a host computer command that performs the download, followed by a null character (CTRL-@ on most terminals) and a carriage return. COM places the object code in 8540 program memory, and puts the program symbols into the symbol table in 8540 system memory.
- c. Log off from your host, and then terminate COM command execution by entering the null character and then pressing the ESC key.

Once you've downloaded the program to the 8540, skip ahead to the heading "Run the Demonstration Program."

CASE 4: PATCH THE PROGRAM INTO MEMORY

This discussion shows you how to patch the demonstration program into 8540 (or 8550) program memory using the P command, and then add the program symbols into the symbol table using the ADDS command.

Ordinarily, you would load the object code and symbols from a binary or hexadecimal load file, as illustrated for Cases 1, 2, and 3. The procedure presented here is **not** normally used for preparing a program for execution. Use this procedure only if you have no standard means for preparing the program, but would still like to try out the demonstration program.

Start Up the 8540

Start up your 8540 and enter the following command to select the 808x emulator:

```
> SEL 8086 <CR>
```

The SEL command automatically sets the emulation mode to 0 and displays the current version number.

Allocate Memory

Enter the following command to allocate a 4K-byte block of program memory to logical addresses 00000--00FFFH.

```
> AL 0 <CR>
```

Zero Out Memory

Before you patch in any code, use the OS/40 F (Fill) command to fill 8540 program memory with zeros. Later, when you examine memory, the zeros make it easy to identify the beginning and end of your code. Enter the following command line to fill memory addresses 40--11F with zeros:

```
> F 40 11F 00 <CR>
```

Check That Memory Was Filled with Zeros. Check the contents of memory with the OS/40 D (Display) command. The display shows the data in hexadecimal format, and also shows the corresponding ASCII characters. Display the contents of memory addresses 40--11F with the following command line:

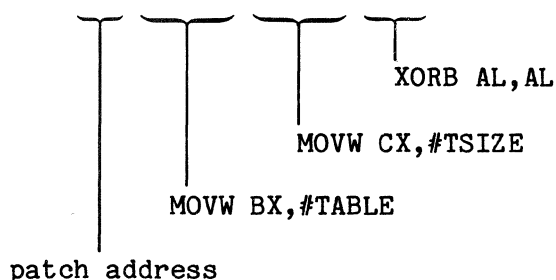
> D 40 11F <CR>

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
000040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000090	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0000A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0000B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0000C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0000D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0000E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0000F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000100	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000110	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

Patch the Object Code into Memory

The OS/40 P (Patch) command stores a sequence of bytes into memory, replacing the previous memory contents. Enter the following command to store the object code for the first three instructions in the program (MOVW BX, TABLE; MOVW CX, TSIZE; and XORB AL,AL) starting at location 100:

> P 100 BB0005 B90500 32C0 <CR>



Now patch in the loop instructions (ADDB AL,[BX]; INCW BX; and LOOP)...

> P 108 020743E2FB <CR>

and now the instructions that call the SVC (MOVW DX, 0FFF7H; OUT DX,AL; and two NOPs):

> P 10D BAF7FFEE9090 <CR>

Finally, patch in the Exit SVC information at address 40:

> P 40 000000441A <CR>

You'll check the contents of memory later in this demonstration.

Put Symbols into the Symbol Table

Later in this demonstration, you will use symbols from the demonstration program (START, LOOP, TSIZE, and TABLE) when communicating with OS/40. Whenever you use a symbol in a command line, OS/40 consults a symbol table in 8540 system memory to find the values that the symbol stands for. Enter the following command line to add the program symbols to the symbol table, along with their values:

```
> ADDS START=100 ALOOP=108 -S TSIZE=5 TABLE=500 <CR>
```

The ADDS command cannot provide all the symbol-related information that is provided by the SYMLO command (as in Cases 1 and 2) or the COM command (as in Case 3). Because this information is missing, some of the displays you produce later in this demonstration will not match the symbolic displays shown in this manual. For more information on the ADDS command, refer to the Command Dictionary of your System Users Manual.

You've patched the demonstration program into program memory and placed the program symbols in the symbol table. Now it's time to run the program.

RUN THE DEMONSTRATION PROGRAM

From now until the end of the demonstration, the commands you are to enter are shown in lowercase. If you are not logged in to an 8560, you may enter commands in either lowercase or uppercase. If you **are** using an 8560, you **must** enter the name of every command in lowercase and your system prompt is "\$", not ">".

Now that you've loaded the program into memory, you need to:

1. Verify that the program was loaded correctly
2. Put values into the table in memory, for the program to add.

Check Memory Contents Again. Before you loaded the program, you filled memory locations 40--11F with zeros. Look at the same memory area again with the following command line:

```
> d 40 11f <CR>
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
000040 00 00 00 44 1A 00 00 00 00 00 00 00 00 00 00 00 ...D.....
000050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000100 BB 00 05 B9 05 00 32 C0 02 07 43 E2 FB BA F7 FF .....2...C.....
000110 EE 90 90 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
```

The object code is loaded in two different blocks:

- The 8086 machine instructions are loaded at address 100 (specified by the first ORG directive in the source code).
- The information for the Exit SVC is loaded at address 40 (specified by the second ORG directive).

The contents of the table at address 500 are still undefined, but you'll put some values into the table in just a few minutes.

Turn On Symbolic Display

Enter the following command to tell the system to modify its displays by replacing hexadecimal numbers with symbols from your program, where appropriate:

> synd on <CR>

Disassemble the Object Code. The DI (Disassemble) command displays memory contents both in hexadecimal notation and in assembly language mnemonics. You can use the DI command to verify that the object code in memory corresponds to your source code. Enter the following command to disassemble the area of memory occupied by the executable part of your program:

```
> di 100 112 <CR>
LOC      INST      MNEM      OPER

START
000100    BB0005      MOVW      BX,#0500

DEMO+000103
000103    B90500      MOVW      CX,#0005

DEMO+000106
000106    32C0        XORB      AL,AL

ALOOP
000108    0207        ADDB      AL,[BX]

DEMO+00010A
00010A    43          INC       BX

DEMO+00010B
00010B    E2FB        LOOP      $-03

DEMO+00010D
00010D    BAF7FF      MOVW      DX,#FFF7

DEMO+000110
000110    EE          OUT       DX,AL

LOC      INST      MNEM      OPER

DEMO+000111
000111    90          NOP

DEMO+000112
000112    90          NOP
```

Compare the DI display with the assembler listing you generated earlier, or refer back to Fig. 7K-5.

The entire memory area used by your program (location 0 through the end of the table --- location 504) belongs to section DEMO. This section was declared by the SECTION directive in the source code. (If you used the ADDS command to create your symbols, as in Case 4, the section name shown in the DI display is NO.SECTION.)

The LOC (location) column of the DI display contains information that enables you to correlate the display with your assembler listing. The symbols START and ALOOP in the DI display correspond to the labels START and ALOOP in the source code. For those lines of the display where the location does not correspond to a label in the symbol table, DI substitutes the address of the instruction **relative to the beginning of the section**, as shown in the address field of your assembler listing. If you haven't loaded the pertinent symbols and related information into the symbol table (using a command such as SYMLO), the DI command supplies only the absolute (actual) addresses in the LOC column. (Since section DEMO begins at address 0, the relative address, or **offset**, is the same as the absolute address in this display. This offset feature is much more useful for sections that **don't** start at address 0.)

Now you've seen that your system can use the symbol table to translate numbers into symbols to make a display easier to read. Your system can also translate a symbol in a command line into an address. For example, since your system knows that the symbol `START` is equivalent to the address 100, you could have entered the `DI` command in any of the following ways:

```
di 100 112
di START 112
di start start+12
di 100 START+12
di start,,10
```

Notice that a symbol can be entered in either lowercase or uppercase.

The feature that enables DOS/50 and OS/40 to correlate symbols from your program with the numbers they represent is termed **symbolic debug**.

Put Values into the Table in Memory. The demonstration program sums five numbers from a table in memory. Use the P (Patch) command to store the numbers 1, 2, 3, 4, and 5 in the table. Do you remember what the address of the table is? It doesn't matter, as long as you remember that the symbol TABLE represents that address.

```
> p table 0102030405 <CR>
```

	
address of table: 500	string of bytes to be stored at addresses 500--504

Check the Contents of the Table. Use the D command to display the contents of the table. (When you don't specify an upper boundary for the area to be displayed, the D command displays 16 bytes.)

7K-76

> g start <CR>

PC/CS:IP	MNEMONIC/DATA	AX/BX	CX/DX	SI/DI	SP/BP	SS/DS	ES/F
DEMO+000112							
000112	NOP	000F	0000	0000	0000	0000	0000
0000:0113	90	0505	FFF7	0000	0000	0000	F006
000112	<BREAK						

The program executes, and when the Exit SVC occurs, the program breaks (stops), and the contents of the emulator registers are displayed. Register AL contains the sum of the numbers in the memory table: 1+2+3+4+5=0F.

MONITOR PROGRAM EXECUTION

You have assembled, loaded, and executed the demonstration program. The rest of this demonstration shows you some commands for monitoring program execution. You can watch the changes in the emulator's registers and observe the effect of each instruction as the program proceeds.

Trace All Instructions. The TRA (TRAcE) command lets you observe the changes in the 8086 registers as the program proceeds. When you enter a TRA command and then start execution with the G command, display lines are sent to the system terminal. As each instruction executes, the display line shows the instruction (as in the DI display) and the contents of the registers after that instruction has executed. Enter the following command to trace all of the program's instructions:

```
> tra all <CR>
```

Enter the command G START (or G 100) to resume program execution back at the beginning of the program:

```
> g start <CR>
```

As the program executes, the following trace is displayed. Remember that you can type CTRL-S to suspend the display and CTRL-Q to resume the display.

PC/CS:IP	MNEMONIC/DATA	AX/BX	CX/DX	SI/DI	SP/BP	SS/DS	ES/F
START							
000100	MOVW BX,#0500	0063	0000	0000	0000	0000	0000
0000:0103	BB0005	0500	FFF7	0000	0000	0000	F006
DEMO+000103							
000103	MOVW CX,#0005	0063	0005	0000	0000	0000	0000
0000:0106	B90500	0500	FFF7	0000	0000	0000	F006
DEMO+000106							
000106	XORB AL,AL	0000	0005	0000	0000	0000	0000
0000:0108	32C0	0500	FFF7	0000	0000	0000	F046
ALOOP							
000108	ADDB AL,[BX]	0001	0005	0000	0000	0000	0000
0000:010A	0207	0500	FFF7	0000	0000	0000	F046
DEMO+00010A							
00010A	INC BX	0001	0005	0000	0000	0000	0000
0000:010B	43	0501	FFF7	0000	0000	0000	F002
PC/CS:IP MNEMONIC/DATA AX/BX CX/DX SI/DI SP/BP SS/DS ES/F							
DEMO+00010B							
00010B	LOOP \$-03	0001	0004	0000	0000	0000	0000
0000:0108	E2FB	0501	FFF7	0000	0000	0000	F002

ALoop									
000108	ADDB	AL,[BX]	0003	0004	0000	0000	0000	0000	
0000:010A	0207		0501	FFF7	0000	0000	0000	F046	
DEMO+00010A									
00010A	INC	BX	0003	0004	0000	0000	0000	0000	
0000:010B	43		0502	FFF7	0000	0000	0000	F002	
DEMO+00010B									
00010B	LOOP	\$-03	0003	0003	0000	0000	0000	0000	
0000:0108	E2FB		0502	FFF7	0000	0000	0000	F002	
ALoop									
000108	ADDB	AL,[BX]	0006	0003	0000	0000	0000	0000	
0000:010A	0207		0502	FFF7	0000	0000	0000	F006	
PC/CS:IP MNEMONIC/DATA				AX/BX	CX/DX	SI/DI	SP/BP	SS/DS	ES/F
DEMO+00010A									
00010A	INC	BX	0006	0003	0000	0000	0000	0000	
0000:010B	43		0503	FFF7	0000	0000	0000	F006	
DEMO+00010B									
00010B	LOOP	\$-03	0006	0002	0000	0000	0000	0000	
0000:0108	E2FB		0503	FFF7	0000	0000	0000	F006	
ALoop									
000108	ADDB	AL,[BX]	000A	0002	0000	0000	0000	0000	
0000:010A	0207		0503	FFF7	0000	0000	0000	F002	
DEMO+00010A									
00010A	INC	BX	000A	0002	0000	0000	0000	0000	
0000:010B	43		0504	FFF7	0000	0000	0000	F002	
DEMO+00010B									
00010B	LOOP	\$-03	000A	0001	0000	0000	0000	0000	
0000:0108	E2FB		0504	FFF7	0000	0000	0000	F002	
PC/CS:IP MNEMONIC/DATA				AX/BX	CX/DX	SI/DI	SP/BP	SS/DS	ES/F
ALoop									
000108	ADDB	AL,[BX]	000F	0001	0000	0000	0000	0000	
0000:010A	0207		0504	FFF7	0000	0000	0000	F016	
DEMO+00010A									
00010A	INC	BX	000F	0001	0000	0000	0000	0000	
0000:010B	43		0505	FFF7	0000	0000	0000	F006	
DEMO+00010B									
00010B	LOOP	\$-03	000F	0000	0000	0000	0000	0000	
0000:010D	E2FB		0505	FFF7	0000	0000	0000	F006	
DEMO+00010D									
00010D	MOVW	DX, FFF7	000F	0000	0000	0000	0000	0000	

```

0000:0110      BAF7FF              0505  FFF7  0000  0000  0000  F006

DEMO+000110
000110      OUT      DX,AL          0000F 0000  0000  0000  0000  0000
0000:0111      EE                  0505  FFF7  0000  0000  0000  F006

000110  <BREAK      TRACE>

```

After the accumulator is cleared, it begins to store the sum of the numbers being added. The `ADDB AL,[BX]` instruction adds a number from the table into the accumulator. At the end of the program, the accumulator contains the sum of the numbers you put into the table.

Register CX, the pass counter, is set to contain 5 (TSIZE) at the beginning of the program. It decreases by one (because of the `LOOP` instruction) each time a number is added into the accumulator. The program ends after register CX reaches zero.

The BX register, set to contain 500 (TABLE) at the start of the program, increases by one each time a number is added to the accumulator. At the end of the program, the BX register has been incremented five times and contains 505.

Trace to the Line Printer. By adding the parameter `>LPT` to a command, you can direct that command's output to the line printer instead of to the system terminal. First, verify that your line printer is properly connected and powered up. Then enter the following command to execute the program and direct the trace output to the line printer:

```
> g start >LPT <CR>
```

NOTE

If you're operating in TERM mode with an 8560, use one of the following commands in place of the command shown:

- `g start | lpr` sends the display to the 8560 line printer.
- `g start \>LPT` sends the display to the line printer on the 8540 or 8550.

Trace Jump Instructions Only. Another way to monitor the program's execution is to look at only the jump instructions. By tracing the jump instructions, you can still observe the changes in the registers, but you save time and space by not tracing the instructions within the loop. Enter the following command to trace only the jump instructions when the loop is being executed:

```
> tra jmp aloop 112 <CR>
```

```

      |
      |
      |----- upper address } Within this range,
      |                     } only jump instructions
      |----- lower address } are traced.
      |                     (108)

```

The TRA command without any parameters displays the trace conditions that are currently set. Because you can have up to three trace selections in effect at the same time, it is useful to be able to see which selections are active. Check your trace status with the following command line:

```
> tra <CR>
TRACE    ALL,000000,00FFFF
TRACE    JMP,ALoop,DEMO+000112
```

As you've specified, TRA ALL is in effect for addresses 0--108, TRA JMP is in effect for addresses 108--112, and TRA ALL is again in effect for addresses 113--FFFF.

Again, start your program with the G command. The following trace is displayed:

```
> g start <CR>
```

PC/CS:IP	MNEMONIC/DATA	AX/BX	CX/DX	SI/DI	SP/BP	SS/DS	ES/F
START							
000100	MOVW BX,#0500	000F	0000	0000	0000	0000	0000
0000:0103	BB0005	0500	FFF7	0000	0000	0000	F006
DEMO+000103							
000103	MOVW CX,#0005	000F	0005	0000	0000	0000	0000
0000:0106	B90500	0500	FFF7	0000	0000	0000	F006
DEMO+000106							
000106	XORB AL,AL	0000	0005	0000	0000	0000	0000
0000:0108	32C0	0500	FFF7	0000	0000	0000	F046
DEMO+00010B							
00010B	LOOP \$-03	0001	0004	0000	0000	0000	0000
0000:0108	E2FB	0501	FFF7	0000	0000	0000	F002
DEMO+00010B							
00010B	LOOP \$-03	0003	0003	0000	0000	0000	0000
0000:0108	E2FB	0502	FFF7	0000	0000	0000	F002
PC/CS:IP MNEMONIC/DATA AX/BX CX/DX SI/DI SP/BP SS/DS ES/F							
DEMO+00010B							
00010B	LOOP \$-03	0006	0002	0000	0000	0000	0000
0000:0108	E2FB	0503	FFF7	0000	0000	0000	F006

7K-82

Use the G command to start program execution:

> g start <CR>

PC/CS:IP	MNEMONIC/DATA	AX/BX	CX/DX	SI/DI	SP/BP	SS/DS	ES/F
START							
000100	MOVW BX,#0500	000F	0000	0000	0000	0000	0000
0000:0103	BB0005	0500	FFF7	0000	0000	0000	F006
DEMO+000103							
000103	MOVW CX,#0005	000F	0005	0000	0000	0000	0000
0000:0106	B90500	0500	FFF7	0000	0000	0000	F006
000103	<BREAK	TRACE, BKPT1>					

The TRA ALL command enabled display of all instructions up to and including the instruction at the breakpoint.

Set New Values in Pass Counter and Table Pointer; Check Results. Now that you've reached the breakpoint, you can change the contents of the registers while execution is stopped. The break display shows that register CX (the pass counter) contains 5, and register BX (the table pointer) contains the address 500. Use the S (Set) command to set the number of passes to two and set the table pointer to 502:

> s bx=502 cx=5 <CR>

The S command does not produce a display, but you can use the DS (Display Status) command to check the values in the registers you changed. DS displays the contents of each emulator register and status flag. Check the result of the previous S command with the following command line:

> <u>ds</u> <CR>		PC/CS:IP	HD STATUS/CLOCK	AX/BX	CX/DX	SI/DI	SP/BP	SS/DS	ES/F																																		
000106		NMI=0	INTR=0	TEST=1	000F	0002	0000	0000	0000																																		
0000:0106		8MHZ			0502	FFF7	0000	0000	F006																																		
<table border="0"> <tr> <td>FLAGS</td> <td>.</td><td>.</td><td>.</td><td>.</td> <td>OF</td><td>DF</td><td>IF</td><td>TF</td> <td>SF</td><td>ZF</td> <td>.</td><td>AF</td> <td>.</td><td>PF</td> <td>.</td><td>CF</td> </tr> <tr> <td>F060</td> <td>X</td><td>X</td><td>X</td><td>X</td> <td>0</td><td>0</td><td>0</td><td>0</td> <td>0</td><td>0</td> <td>X</td><td>0</td> <td>X</td><td>1</td> <td>X</td><td>0</td> </tr> </table>										FLAGS	.	.	.	.	OF	DF	IF	TF	SF	ZF	.	AF	.	PF	.	CF	F060	X	X	X	X	0	0	0	0	0	0	X	0	X	1	X	0
FLAGS	.	.	.	.	OF	DF	IF	TF	SF	ZF	.	AF	.	PF	.	CF																											
F060	X	X	X	X	0	0	0	0	0	0	X	0	X	1	X	0																											

The DS display shows that the pass counter and table pointer now contain the new values.

Resume Program Execution. If you enter the G command with no parameters, program execution starts where it left off. Resume program execution after the breakpoint with the following command:

> g <CR>

PC/CS:IP	MNEMONIC/DATA	AX/BX	CX/DX	SI/DI	SP/BP	SS/DS	ES/F
DEMO+000106							
000106	XORB AL,AL	0000	0002	0000	0000	0000	0000
0000:0108	32C0	0502	FFF7	0000	0000	0000	F046
ALOOP							
000108	ADDB AL,[BX]	0003	0002	0000	0000	0000	0000
0000:010A	0207	0502	FFF7	0000	0000	0000	F006
DEMO+00010A							
00010A	INC BX	0003	0002	0000	0000	0000	0000
0000:010B	43	0503	FFF7	0000	0000	0000	F006
DEMO+00010B							
00010B	LOOP \$-03	0003	0001	0000	0000	0000	0000
0000:0108	E2FB	0503	FFF7	0000	0000	0000	F006
ALOOP							
000108	ADDB AL,[BX]	0007	0001	0000	0000	0000	0000
0000:010A	0207	0503	FFF7	0000	0000	0000	F002
PC/CS:IP	MNEMONIC/DATA	AX/BX	CX/DX	SI/DI	SP/BP	SS/DS	ES/F
DEMO+00010A							
00010A	INC BX	0007	0001	0000	0000	0000	0000
0000:010B	43	0504	FFF7	0000	0000	0000	F002
DEMO+00010B							
00010B	LOOP \$-03	0007	0000	0000	0000	0000	0000
0000:010D	E2FB	0504	FFF7	0000	0000	0000	F002
DEMO+00010D							
00010D	MOVW DX, FFF7	0007	0000	0000	0000	0000	0000
0000:0110	BAF7FF	0504	FFF7	0000	0000	0000	F002
DEMO+000110							
000110	OUT DX,AL	0007	0000	0000	0000	0000	0000
0000:0111	EE	0504	FFF7	0000	0000	0000	F002
000110	<BREAK TRACE>						

Notice that the program performed two passes through the loop, and that the program added the third and fourth numbers in the table: 3+4=7.

SUMMARY OF 8086 EMULATOR DEMONSTRATION RUN

You have assembled, loaded, executed, and monitored the demonstration run program. Review the commands you used:

- **SEL---**selects the 8086/8088 assembler and emulator
- **ASM---**creates object code from an assembly language program
- **LINK---**links object code into a load module
- **F---**fills an area of memory with a specified value
- **D---**displays memory contents in ASCII and hexadecimal format
- **LO---**loads object code into memory
- **SYMLO---**loads program symbols for use in symbolic debug
- **DI---**translates memory contents into assembly language mnemonics
- **P---**patches a string of bytes into memory
- **SYMD---**enables symbolic debug display
- **G---**begins or resumes program execution
- **TRA---**selects instructions to be traced during program execution
- **BK---**sets a breakpoint
- **S---**modifies emulator registers
- **DS---**displays emulator registers

Delete the Demonstration Run Files

Now that you've finished the demonstration run, you can delete the source file, object file, listing file, and load file. If you're using an 8550, the source and load files are still available to you on the 8086/8087/8088 emulator installation disk. If you're using an 8560, remember that once you delete the source file (asm), there is no way of recovering it.

Delete 8550 Files. If your files are on the 8550, use the following procedure to delete them. First use the USER command to move from the DEMO directory back into the directory you were in at the start of the demonstration. Recall that you marked that directory with the brief name /ROOT.

> USER /ROOT <CR>

Now enter the following command to delete the DEMO directory and the files it contains:

```
> DEL DEMO/* DEMO <CR>
Delete ASM ? Y <CR>
Delete LOAD ? Y <CR>
Delete OBJ ? Y <CR>
Delete ASML ? Y <CR>
Delete DEMO ? Y <CR>
```

Before deleting each file, DOS/50 asks you whether you really want to delete it. You type "Y" for yes.

Delete 8560 Files. If your files are on the 8560, use the following procedure to delete them. Enter the following command to remove all files in the working directory, including the source file:

```
$ rm * <CR>
```

Now move from the demo directory back into the parent directory and remove the demo directory itself:

```
$ cd .. <CR>
$ rmdir demo <CR>
```

Turn Off Your System

For instructions on turning off your 8540 or 8550, refer to the Learning Guide of your System Users Manual.

MANUAL CHANGE INFORMATION

At Tektronix, we continually strive to keep up with latest electronic developments by adding circuit and component improvements to our instruments as soon as they are developed and tested.

Sometimes, due to printing and shipping requirements, we can't get these changes immediately into printed manuals. Hence, your manual may contain new change information on following pages.

A single change may affect several sections. Since the change information sheets are carried in the manual until all changes are permanently entered, some duplication may occur. If no such change pages appear following this page, your manual is correct as printed.